

Intel® IXP45X and Intel® IXP46X Product Line of Network Processors

Developer's Manual

August 2006



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel products are not intended for use in medical, life-saving, life-sustaining, critical control or safety systems, or in nuclear-facility applications. Intel may make changes to specifications and product descriptions at any time, without notice.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® IXP45X and Intel® IXP46X Product Line of Network Processors may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Intel, Intel logo, and Intel XScale are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2006, Intel Corporation



Contents

1.0	Introduction.....	38
1.1	About This Document.....	38
1.2	Intended Audience	38
1.3	How to Read This Document	38
1.4	Other Relevant Documents	38
1.5	Terminology and Conventions	39
1.5.1	Number Representation.....	42
1.5.2	Signal-Naming Convention	42
1.5.3	Register Legend	43
2.0	Functional Overview	44
2.1	Key Functional Units	48
2.1.1	Network Processor Engines (NPEs).....	48
2.1.2	Internal Bus	49
2.1.2.1	North AHB	49
2.1.2.2	South AHB	50
2.1.2.3	Memory Port Interface.....	50
2.1.2.4	APB Bus	51
2.1.3	MII/SMII Interfaces	51
2.1.4	UTOPIA Level 2	51
2.1.5	Universal Serial Bus (USB) Interfaces.....	52
2.1.5.1	USB 1.1 Device Interface	52
2.1.5.2	USB 2.0 Host Interface	52
2.1.6	PCI Controller	52
2.1.7	DDR1 SDRAM Controller	53
2.1.8	Expansion Interface	54
2.1.9	High-Speed, Serial Interfaces.....	56
2.1.10	UARTs	56
2.1.11	GPIO	56
2.1.12	Internal Bus Performance Monitoring Unit (IBPMU)	57
2.1.13	Interrupt Controller	57
2.1.14	Timers	58
2.1.15	IEEE-1588 Hardware Assist	58
2.1.16	Synchronous Serial Protocol Interface	58
2.1.17	I2C Interface	59
2.1.18	AES/DES/SHA/MD-5	59
2.1.19	Cryptography Unit	59
2.1.20	Queue Manager.....	60
2.2	Intel XScale® Processor	60
2.2.1	Super Pipeline.....	62
2.2.2	Branch Target Buffer	63
2.2.3	Instruction Memory Management Unit.....	63
2.2.4	Data Memory Management Unit	64
2.2.5	Instruction Cache	64
2.2.6	Data Cache	65
2.2.7	Mini-Data Cache.....	65
2.2.8	Fill Buffer and Pend Buffer	65
2.2.9	Write Buffer.....	66
2.2.10	Multiply-Accumulate Coprocessor	66
2.2.11	Performance Monitoring Unit	66
2.2.12	Debug Unit.....	67
3.0	Intel XScale® Processor	69



3.1	Memory Management Unit	69
3.1.1	Memory Attributes	70
3.1.1.1	Page (P) Attribute Bit	70
3.1.1.2	Cacheable (C), Bufferable (B), and eXtension (X) Bits	70
3.1.2	Interaction of the MMU, Instruction Cache, and Data Cache	72
3.1.3	MMU Control	73
3.1.3.1	Invalidate (Flush) Operation	73
3.1.3.2	Enabling/Disabling	73
3.1.3.3	Locking Entries	74
3.1.3.4	Round-Robin Replacement Algorithm	76
3.2	Instruction Cache	77
3.2.1	Operation When Instruction Cache is Enabled	77
3.2.1.1	Instruction-Cache 'Miss'	78
3.2.1.2	Instruction-Cache Line-Replacement Algorithm	79
3.2.1.3	Instruction-Cache Coherence	80
3.3	Branch Target Buffer	83
3.3.1	Branch Target Buffer (BTB) Operation	83
3.3.1.1	Reset	85
3.4	Data Cache	85
3.4.1	Data Cache Overview	85
3.4.2	Cacheability	88
3.4.3	Reconfiguring the Data Cache as Data RAM	92
3.5	Configuration	96
3.5.1	CP15 Registers	98
3.5.1.1	Register 0: ID & Cache Type Registers	98
3.5.1.2	Register 1: Control and Auxiliary Control Registers	100
3.5.1.3	Register 2: Translation Table Base Register	102
3.5.1.4	Register 3: Domain Access Control Register	102
3.5.1.5	Register 4: Reserved	102
3.5.1.6	Register 5: Fault Status Register	102
3.5.1.7	Register 6: Fault Address Register	103
3.5.1.8	Register 7: Cache Functions	103
3.5.1.9	Register 8: TLB Operations	104
3.5.1.10	Register 9: Cache Lock Down	105
3.5.1.11	Register 10: TLB Lock Down	106
3.5.1.12	Register 11-12: Reserved	106
3.5.1.13	Register 13: Process ID	106
3.5.1.14	The PID Register Affect On Addresses	107
3.5.1.15	Register 14: Breakpoint Registers	107
3.5.1.16	Register 15: Coprocessor Access Register	107
3.5.2	CP14 Registers	108
3.5.2.1	Performance Monitoring Registers	109
3.5.2.2	Clock and Power Management Registers	109
3.5.2.3	Software Debug Registers	110
3.6	Software Debug	111
3.6.1	Definitions	111
3.6.2	Debug Registers	111
3.6.3	Debug Modes	112
3.6.3.1	Halt Mode	112
3.6.3.2	Monitor Mode	112
3.6.4	Debug Control and Status Register (DCSR)	112
3.6.4.1	Global Enable Bit (GE)	114
3.6.4.2	Halt Mode Bit (H)	114
3.6.4.3	Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR)	114
3.6.4.4	Sticky Abort Bit (SA)	114
3.6.4.5	Method of Entry Bits (MOE)	114
3.6.4.6	Trace Buffer Mode Bit (M)	114



3.6.4.7	Trace Buffer Enable Bit (E)	114
3.6.5	Debug Exceptions.....	115
3.6.5.1	Halt Mode	115
3.6.5.2	Monitor Mode	117
3.6.6	HW Breakpoint Resources.....	117
3.6.6.1	Instruction Breakpoints.....	118
3.6.6.2	Data Breakpoints	118
3.6.7	Software Breakpoints.....	120
3.6.8	Transmit/Receive Control Register	120
3.6.8.1	RX Register Ready Bit (RR)	121
3.6.8.2	Overflow Flag (OV)	122
3.6.8.3	Download Flag (D)	122
3.6.8.4	TX Register Ready Bit (TR).....	123
3.6.8.5	Conditional Execution Using TXRXCTRL	123
3.6.9	Transmit Register	124
3.6.10	Receive Register	124
3.6.11	Debug JTAG Access	124
3.6.11.1	SELDCSR JTAG Command	125
3.6.11.2	SELDCSR JTAG Register.....	125
3.6.11.3	DBGTX JTAG Command	127
3.6.11.4	DBGTX JTAG Register	127
3.6.11.5	DBGTX JTAG Command	128
3.6.11.6	DBGTX JTAG Register	128
3.6.11.7	Debug JTAG Data Register Reset Values.....	132
3.6.12	Trace Buffer	132
3.6.12.1	Trace Buffer CP Registers.....	132
3.6.13	Trace Buffer Entries.....	134
3.6.13.1	Message Byte	134
3.6.13.2	Trace Buffer Usage.....	137
3.6.14	Downloading Code in ICache.....	139
3.6.14.1	LDIC JTAG Command	139
3.6.14.2	LDIC JTAG Data Register	140
3.6.14.3	LDIC Cache Functions.....	141
3.6.14.4	Loading IC During Reset	142
3.6.14.5	Dynamically Loading IC After Reset	147
3.6.14.6	Mini-Instruction Cache Overview	150
3.6.15	Halt Mode Software Protocol	150
3.6.15.1	Starting a Debug Session.....	150
3.6.15.2	Implementing a Debug Handler	152
3.6.15.3	Ending a Debug Session	155
3.6.16	Software Debug Notes and Errata.....	156
3.7	Performance Monitoring	157
3.7.1	Overview	157
3.7.2	Register Description.....	158
3.7.2.1	Clock Counter (CCNT)	158
3.7.2.2	Performance Count Registers.....	158
3.7.2.3	Performance Monitor Control Register	159
3.7.2.4	Interrupt Enable Register.....	159
3.7.2.5	Overflow Flag Status Register	160
3.7.2.6	Event Select Register	161
3.7.3	Managing the Performance Monitor	162
3.7.4	Performance Monitoring Events	162
3.7.4.1	Instruction Cache Efficiency Mode	163
3.7.4.2	Data Cache Efficiency Mode.....	163
3.7.4.3	Instruction Fetch Latency Mode	164
3.7.4.4	Data/Bus Request Buffer Full Mode.....	164
3.7.4.5	Stall/Write-Back Statistics.....	165



3.7.4.6	Instruction TLB Efficiency Mode	165
3.7.4.7	Data TLB Efficiency Mode	165
3.7.5	Multiple Performance Monitoring Run Statistics	166
3.7.6	Examples.....	166
3.8	Programming Model	167
3.8.1	Intel® StrongARM® Architecture Compatibility.....	167
3.8.2	Intel® StrongARM® Architecture Implementation Options	168
3.8.2.1	Big-Endian versus Little-Endian	168
3.8.2.2	26-Bit Architecture	168
3.8.2.3	Thumb.....	168
3.8.2.4	Intel® StrongARM® DSP-Enhanced Instruction Set	168
3.8.2.5	Base Register Update.....	169
3.8.3	Extensions to Intel® StrongARM® Architecture	169
3.8.3.1	DSP Coprocessor 0 (CP0)	169
3.8.3.2	New Page Attributes	175
3.8.3.3	Additions to CP15 Functionality	176
3.8.3.4	Event Architecture.....	177
3.9	Performance Considerations	181
3.9.1	Interrupt Latency.....	181
3.9.2	Branch Prediction.....	182
3.9.3	Addressing Modes.....	182
3.9.4	Instruction Latencies.....	182
3.9.4.1	Performance Terms	182
3.9.4.2	Branch Instruction Timings	184
3.9.4.3	Data Processing Instruction Timings.....	184
3.9.4.4	Multiply Instruction Timings.....	185
3.9.4.5	Saturated Arithmetic Instructions	187
3.9.4.6	Status Register Access Instructions.....	187
3.9.4.7	Load/Store Instructions.....	187
3.9.4.8	Semaphore Instructions	188
3.9.4.9	Coprocessor Instructions	188
3.9.4.10	Miscellaneous Instruction Timing	189
3.9.4.11	Thumb Instructions	189
3.10	Optimization Guide	189
3.10.1	Introduction.....	189
3.10.1.1	About This Section	190
3.10.2	Processor Pipeline	190
3.10.2.1	General Pipeline Characteristics.....	190
3.10.2.2	Instruction Flow Through the Pipeline.....	192
3.10.2.3	Main Execution Pipeline	193
3.10.2.4	Memory Pipeline.....	194
3.10.2.5	Multiply/Multiply Accumulate (MAC) Pipeline	195
3.10.3	Basic Optimizations.....	195
3.10.3.1	Conditional Instructions	195
3.10.3.2	Bit Field Manipulation.....	199
3.10.3.3	Optimizing the Use of Immediate Values	200
3.10.3.4	Optimizing Integer Multiply and Divide	200
3.10.3.5	Effective Use of Addressing Modes	201
3.10.4	Cache and Prefetch Optimizations	201
3.10.4.1	Instruction Cache	201
3.10.4.2	Data and Mini Cache	203
3.10.4.3	Cache Considerations.....	206
3.10.4.4	Prefetch Considerations.....	207
3.10.5	Instruction Scheduling.....	212
3.10.5.1	Scheduling Loads	212
3.10.5.2	Scheduling Data Processing Instructions.....	216
3.10.5.3	Scheduling Multiply Instructions	217
3.10.5.4	Scheduling SWP and SWPB Instructions	218



3.10.5.5	Scheduling the MRA and MAR Instructions (MRRC/MCRR)	218
3.10.5.6	Scheduling the MIA and MIAPH Instructions	219
3.10.5.7	Scheduling MRS and MSR Instructions	219
3.10.5.8	Scheduling CP15 Coprocessor Instructions	220
3.10.6	Optimizing C Libraries	220
3.10.7	Optimizations for Size	220
3.10.7.1	Space/Performance Trade Off	220
4.0	Network Processor Engines (NPE)	222
5.0	Internal Buses	224
5.1	Internal Bus Arbiters	225
5.1.1	Priority Mechanism	225
5.2	Memory Map	226
6.0	Ethernet MACs	229
6.1	Ethernet Coprocessor	230
6.1.1	Ethernet Coprocessor APB Interface	230
6.1.2	Ethernet Coprocessor NPE Interface	230
6.1.3	Ethernet Coprocessor MDIO Interface	231
6.1.4	Transmitting Ethernet Frames with MII Interfaces	233
6.1.5	Receiving Ethernet Frames with MII Interfaces	236
6.1.6	General Ethernet Coprocessor Configuration	238
6.2	Register Descriptions Ethernet MACs	239
6.2.1	Ethernet MAC 0 on NPE B	240
6.2.2	Ethernet MAC 1 on NPE B	241
6.2.3	Ethernet MAC 2 on NPE B	243
6.2.4	Ethernet MAC 2 on NPE B	244
6.2.5	Ethernet MAC on NPE A	246
6.2.6	Ethernet MAC on NPE C	247
6.2.7	Transmit Control 1	249
6.2.8	Transmit Control 2	249
6.2.9	Receive Control 1	250
6.2.10	Receive Control 2	251
6.2.11	Random Seed	251
6.2.12	Threshold For Partially Empty	251
6.2.13	Threshold For Partially Full	252
6.2.14	Buffer Size For Transmit	252
6.2.15	Transmit Deferral Parameter	253
6.2.16	Receive Deferral Parameter	253
6.2.17	Transmit Two Part Deferral Parameters 1	253
6.2.18	Transmit Two Part Deferral Parameters 2	254
6.2.19	Slot Time	254
6.2.20	MDIO Commands Registers	254
6.2.21	MDIO Command 1	255
6.2.22	MDIO Command 2	255
6.2.23	MDIO Command 3	255
6.2.24	MDIO Command 4	256
6.2.25	MDIO Status Registers	256
6.2.26	MDIO Status 1	256
6.2.27	MDIO Status 2	257
6.2.28	MDIO Status 3	257
6.2.29	MDIO Status 4	257
6.2.30	Address Mask Registers	257
6.2.31	Address Mask 1	258
6.2.32	Address Mask 2	258



6.2.33	Address Mask 3	259
6.2.34	Address Mask 4	259
6.2.35	Address Mask 5	259
6.2.36	Address Mask 6	260
6.2.37	Address Registers	260
6.2.38	Address 1	261
6.2.39	Address 2	261
6.2.40	Address 3	261
6.2.41	Address 4	262
6.2.42	Address 5	262
6.2.43	Address 6	262
6.2.44	Threshold for Internal Clock	263
6.2.45	Unicast Address Registers	263
6.2.46	Unicast Address 1	264
6.2.47	Unicast Address 2	264
6.2.48	Unicast Address 3	264
6.2.49	Unicast Address 4	264
6.2.50	Unicast Address 5	265
6.2.51	Unicast Address 6	265
6.2.52	Core Control	265
7.0	UTOPIA Level 2	268
7.1	Introduction	268
7.2	UTOPIA Transmit Module	269
7.3	UTOPIA Receive Module	272
7.4	UTOPIA Level 2 Coprocessor / NPE Coprocessor: Bus Interface	275
7.5	MPHY Polling Routines	275
7.6	UTOPIA Level 2 Clocks.....	276
8.0	USB 1.1 Device Controller.....	278
8.1	USB Overview	278
8.2	Device Configuration	279
8.3	USB Operation	280
8.3.1	Signalling Levels.....	280
8.3.2	Bit Encoding.....	281
8.3.3	Field Formats	282
8.3.4	Packet Formats	283
8.3.4.1	Token Packet Type	284
8.3.4.2	Start-of-Frame Packet Type	284
8.3.4.3	Data Packet Type	284
8.3.4.4	Handshake Packet Type	285
8.3.5	Transaction Formats.....	285
8.3.5.1	Bulk Transaction Type	285
8.3.5.2	Isochronous Transaction Type	286
8.3.5.3	Control Transaction Type	286
8.3.5.4	Interrupt Transaction Type	287
8.3.6	UDC Device Requests	287
8.3.7	UDC Configuration	288
8.4	UDC Hardware Connections.....	289
8.4.1	Self-Powered Device	289
8.4.2	Bus-Powered Devices	289
8.5	Register Descriptions.....	289
8.5.1	UDC Control Register	291
8.5.1.1	UDC Enable	291
8.5.1.2	UDC Active	292
8.5.1.3	UDC Resume (RSM).....	292



8.5.1.4	Resume Interrupt Request (RESIR)	292
8.5.1.5	Suspend Interrupt Request (SUSIR)	292
8.5.1.6	Suspend/Resume Interrupt Mask (SRM)	292
8.5.1.7	Reset Interrupt Request (RSTIR)	292
8.5.1.8	Reset Interrupt Mask (REM)	292
8.5.2	UDC Endpoint 0 Control/Status Register	293
8.5.2.1	OUT Packet Ready (OPR)	294
8.5.2.2	IN Packet Ready (IPR)	294
8.5.2.3	Flush Tx FIFO (FTF)	294
8.5.2.4	Device Remote Wake-Up Feature (DRWF)	294
8.5.2.5	Sent Stall (SST)	294
8.5.2.6	Force Stall (FST)	295
8.5.2.7	Receive FIFO Not Empty (RNE)	295
8.5.2.8	Setup Active (SA)	295
8.5.3	UDC Endpoint 1 Control/Status Register	296
8.5.3.1	Transmit FIFO Service (TFS)	296
8.5.3.2	Transmit Packet Complete (TPC)	296
8.5.3.3	Flush Tx FIFO (FTF)	296
8.5.3.4	Transmit Underrun (TUR)	296
8.5.3.5	Sent STALL (SST)	297
8.5.3.6	Force STALL (FST)	297
8.5.3.7	Bit 6 Reserved	297
8.5.3.8	Transmit Short Packet (TSP)	297
8.5.4	UDC Endpoint 2 Control/Status Register	298
8.5.4.1	Receive FIFO Service (RFS)	298
8.5.4.2	Receive Packet Complete (RPC)	298
8.5.4.3	Bit 2 Reserved	298
8.5.4.4	Bit 2 Reserved	298
8.5.4.5	Sent Stall (SST)	299
8.5.4.6	Force Stall (FST)	299
8.5.4.7	Receive FIFO Not Empty (RNE)	299
8.5.4.8	Receive Short Packet (RSP)	299
8.5.5	UDC Endpoint 3 Control/Status Register	300
8.5.5.1	Transmit FIFO Service (TFS)	300
8.5.5.2	Transmit Packet Complete (TPC)	301
8.5.5.3	Flush Tx FIFO (FTF)	301
8.5.5.4	Transmit Underrun (TUR)	301
8.5.5.5	Bit 4 Reserved	301
8.5.5.6	Bit 5 Reserved	301
8.5.5.7	Bit 6 Reserved	301
8.5.5.8	Transmit Short Packet (TSP)	301
8.5.6	UDC Endpoint 4 Control/Status Register	302
8.5.6.1	Receive FIFO Service (RFS)	302
8.5.6.2	Receive Packet Complete (RPC)	303
8.5.6.3	Receive Overflow (ROF)	303
8.5.6.4	Bit 3 Reserved	303
8.5.6.5	Bit 4 Reserved	303
8.5.6.6	Bit 5 Reserved	303
8.5.6.7	Receive FIFO Not Empty (RNE)	303
8.5.6.8	Receive Short Packet (RSP)	303
8.5.7	UDC Endpoint 5 Control/Status Register	304
8.5.7.1	Transmit FIFO Service (TFS)	304
8.5.7.2	Transmit Packet Complete (TPC)	305
8.5.7.3	Flush Tx FIFO (FTF)	305
8.5.7.4	Transmit Underrun (TUR)	305
8.5.7.5	Sent STALL (SST)	305
8.5.7.6	Force STALL (FST)	305
8.5.7.7	Bit 6 Reserved	306
8.5.7.8	Transmit Short Packet (TSP)	306



8.5.8	UDC Endpoint 6 Control/Status Register	307
8.5.8.1	Transmit FIFO Service (TFS)	307
8.5.8.2	Transmit Packet Complete (TPC)	307
8.5.8.3	Flush Tx FIFO (FTF)	307
8.5.8.4	Transmit Underrun (TUR)	307
8.5.8.5	Sent STALL (SST).....	307
8.5.8.6	Force STALL (FST).....	308
8.5.8.7	Bit 6 Reserved	308
8.5.8.8	Transmit Short Packet (TSP)	308
8.5.9	UDC Endpoint 7 Control/Status Register.....	309
8.5.9.1	Receive FIFO Service (RFS)	309
8.5.9.2	Receive Packet Complete (RPC)	309
8.5.9.3	Bit 2 Reserved	309
8.5.9.4	Bit 3 Reserved	309
8.5.9.5	Sent Stall (SST)	309
8.5.9.6	Force Stall (FST)	310
8.5.9.7	Receive FIFO Not Empty (RNE).....	310
8.5.9.8	Receive Short Packet (RSP)	310
8.5.10	UDC Endpoint 8 Control/Status Register.....	311
8.5.10.1	Transmit FIFO Service (TFS)	311
8.5.10.2	Transmit Packet Complete (TPC)	311
8.5.10.3	Flush Tx FIFO (FTF)	312
8.5.10.4	Transmit Underrun (TUR)	312
8.5.10.5	Bit 4 Reserved	312
8.5.10.6	Bit 5 Reserved	312
8.5.10.7	Bit 6 Reserved	312
8.5.10.8	Transmit Short Packet (TSP)	312
8.5.11	UDC Endpoint 9 Control/Status Register.....	313
8.5.11.1	Receive FIFO Service (RFS)	313
8.5.11.2	Receive Packet Complete (RPC).....	313
8.5.11.3	Receive Overflow (ROF)	313
8.5.11.4	Bit 3 Reserved	313
8.5.11.5	Bit 4 Reserved	314
8.5.11.6	Bit 5 Reserved	314
8.5.11.7	Receive FIFO Not Empty (RNE).....	314
8.5.11.8	Receive Short Packet (RSP)	314
8.5.12	UDC Endpoint 10 Control/Status Register.....	315
8.5.12.1	Transmit FIFO Service (TFS)	315
8.5.12.2	Transmit Packet Complete (TPC)	315
8.5.12.3	Flush Tx FIFO (FTF)	315
8.5.12.4	Transmit Underrun (TUR)	315
8.5.12.5	Sent STALL (SST).....	316
8.5.12.6	Force STALL (FST).....	316
8.5.12.7	Bit 6 Reserved	316
8.5.12.8	Transmit Short Packet (TSP)	316
8.5.13	UDC Endpoint 11 Control/Status Register.....	317
8.5.13.1	Transmit FIFO Service (TFS)	317
8.5.13.2	Transmit Packet Complete (TPC)	317
8.5.13.3	Flush Tx FIFO (FTF)	318
8.5.13.4	Transmit Underrun (TUR)	318
8.5.13.5	Sent STALL (SST).....	318
8.5.13.6	Force STALL (FST).....	318
8.5.13.7	Bit 6 Reserved	318
8.5.13.8	Transmit Short Packet (TSP)	318
8.5.14	UDC Endpoint 12 Control/Status Register.....	319
8.5.14.1	Receive FIFO Service (RFS)	319
8.5.14.2	Receive Packet Complete (RPC).....	320
8.5.14.3	Bit 2 Reserved	320
8.5.14.4	Bit 3 Reserved	320



8.5.14.5	Sent Stall (SST).....	320
8.5.14.6	Force Stall (FST).....	320
8.5.14.7	Receive FIFO Not Empty (RNE)	320
8.5.14.8	Receive Short Packet (RSP).....	321
8.5.15	UDC Endpoint 13 Control/Status Register	321
8.5.15.1	Transmit FIFO Service (TFS).....	322
8.5.15.2	Transmit Packet Complete (TPC)	322
8.5.15.3	Flush Tx FIFO (FTF).....	322
8.5.15.4	Transmit Underrun (TUR).....	322
8.5.15.5	Bit 4 Reserved.....	322
8.5.15.6	Bit 5 Reserved.....	322
8.5.15.7	Bit 6 Reserved.....	322
8.5.15.8	Transmit Short Packet (TSP)	322
8.5.16	UDC Endpoint 14 Control/Status Register	323
8.5.16.1	Receive FIFO Service (RFS).....	323
8.5.16.2	Receive Packet Complete (RPC)	324
8.5.16.3	Receive Overflow (ROF).....	324
8.5.16.4	Bit 3 Reserved.....	324
8.5.16.5	Bit 4 Reserved.....	324
8.5.16.6	Bit 5 Reserved.....	324
8.5.16.7	Receive FIFO Not Empty (RNE)	324
8.5.16.8	Receive Short Packet (RSP).....	324
8.5.17	UDC Endpoint 15 Control/Status Register	325
8.5.17.1	Transmit FIFO Service (TFS).....	325
8.5.17.2	Transmit Packet Complete (TPC)	326
8.5.17.3	Flush Tx FIFO (FTF).....	326
8.5.17.4	Transmit Underrun (TUR).....	326
8.5.17.5	Sent STALL (SST)	326
8.5.17.6	Force STALL (FST)	326
8.5.17.7	Bit 6 Reserved.....	327
8.5.17.8	Transmit Short Packet (TSP)	327
8.5.18	UDC Interrupt Control Register 0.....	328
8.5.18.1	Interrupt Mask Endpoint x (IMx), where x is 0 through 7.....	328
8.5.19	UDC Interrupt Control Register 1.....	329
8.5.19.1	Interrupt Mask Endpoint x (IMx), where x is 8 through 15.....	329
8.5.20	UDC Status/Interrupt Register 0.....	330
8.5.20.1	Endpoint 0 Interrupt Request (IR0)	330
8.5.20.2	Endpoint 1 Interrupt Request (IR1)	331
8.5.20.3	Endpoint 2 Interrupt Request (IR2)	331
8.5.20.4	Endpoint 3 Interrupt Request (IR3)	331
8.5.20.5	Endpoint 4 Interrupt Request (IR4)	331
8.5.20.6	Endpoint 5 Interrupt Request (IR5)	331
8.5.20.7	Endpoint 6 Interrupt Request (IR6)	331
8.5.20.8	Endpoint 7 Interrupt Request (IR7)	331
8.5.21	UDC Status/Interrupt Register 1.....	332
8.5.21.1	Endpoint 8 Interrupt Request (IR8)	332
8.5.21.2	Endpoint 9 Interrupt Request (IR9)	332
8.5.21.3	Endpoint 10 Interrupt Request (IR10).....	333
8.5.21.4	Endpoint 11 Interrupt Request (IR11).....	333
8.5.21.5	Endpoint 12 Interrupt Request (IR12).....	333
8.5.21.6	Endpoint 13 Interrupt Request (IR13).....	333
8.5.21.7	Endpoint 14 Interrupt Request (IR14).....	333
8.5.21.8	Endpoint 15 Interrupt Request (IR15).....	333
8.5.22	UDC Frame Number High Register	334
8.5.22.1	UDC Frame Number MSB (FNMSB)	334
8.5.22.2	Isochronous Packet Error Endpoint 4 (IPE4)	335
8.5.22.3	Isochronous Packet Error Endpoint 9 (IPE9)	335
8.5.22.4	Isochronous Packet Error Endpoint 14 (IPE14)	335



8.5.22.5	Start of Frame Interrupt Mask (SIM)	335
8.5.22.6	Start of Frame Interrupt Request (SIR)	335
8.5.23	UDC Frame Number Low Register	336
8.5.24	UDC Byte Count Register 2	337
8.5.24.1	Endpoint 2 Byte Count (BC[7:0])	337
8.5.25	UDC Byte Count Register 4	338
8.5.25.1	Endpoint 4 Byte Count (BC[7:0])	338
8.5.26	UDC Byte Count Register 7	338
8.5.26.1	Endpoint 7 Byte Count (BC[7:0])	339
8.5.27	UDC Byte Count Register 9	339
8.5.27.1	Endpoint 9 Byte Count (BC[7:0])	339
8.5.28	UDC Byte Count Register 12	340
8.5.28.1	Endpoint 12 Byte Count (BC[7:0])	340
8.5.29	UDC Byte Count Register 14	341
8.5.29.1	Endpoint 14 Byte Count (BC[7:0])	341
8.5.30	UDC Endpoint 0 Data Register	341
8.5.31	UDC Data Register 1	342
8.5.32	UDC Data Register 2	343
8.5.33	UDC Data Register 3	343
8.5.34	UDC Data Register 4	344
8.5.35	UDC Data Register 5	345
8.5.36	UDC Data Register 6	345
8.5.37	UDC Data Register 7	346
8.5.38	UDC Data Register 8	346
8.5.39	UDC Data Register 9	347
8.5.40	UDC Data Register 10	348
8.5.41	UDC Data Register 11	348
8.5.42	UDC Data Register 12	349
8.5.43	UDC Data Register 13	349
8.5.44	UDC Data Register 14	350
8.5.45	UDC Data Register 15	351
9.0	USB 2.0 Host Controller	352
9.1	Overview	352
9.2	USB	352
9.3	USB 2.0	353
9.4	Feature List	354
9.5	Block Diagram	355
9.6	Theory of Operation	355
9.6.1	Software Model	355
9.6.2	Host Data Structure	355
9.6.3	Hardware Model	358
9.6.3.1	Block Diagram	358
9.6.3.2	Microprocessor Interface	359
9.6.3.3	DMA Engine	360
9.6.3.4	Dual Port RAM Controller	361
9.6.3.5	Protocol Engine	361
9.6.3.6	Port Controller	362
9.6.3.7	System Bus Interface	363
9.7	System Level Issues and Core Configuration	363
9.7.1	Configuration Constants	363
9.8	Detailed Register Descriptions	364
9.9	Configuration, Control and Status Register Set	366
9.10	Identification Registers	367
9.10.1	ID	367
9.10.2	HWGENERAL	367



9.10.3	HWHOST	368
9.10.4	HWDEVICE	368
9.10.5	HWTXBUF	369
9.10.6	HWRXBUF	369
9.11	Host Capability Registers	370
9.11.1	CAPLENGTH – EHCI Compliant	370
9.11.2	HCVERSION – EHCI Compliant	370
9.11.3	HCSPARAMS – EHCI Compliant with Extensions	370
9.11.4	HCCPARAMS – EHCI Compliant	371
9.11.5	Reserved Register #1	372
9.11.6	DCCPARAMS (Non-EHCI)	372
9.12	Host Operational Registers	373
9.12.1	USBCMD	373
9.12.2	USBSTS	375
9.12.3	USBINTR	377
9.12.4	FRINDEX	378
9.12.5	CTRLDSSEGMENT	379
9.12.6	PERIODICLISTBASE	379
9.12.6.1	Host Controller (PERIODICLISTBASE)	379
9.12.7	ASYNCLISTADDR; ENDPOINTLISTADDR	380
9.12.7.1	Host Controller (ASYNCLISTADDR)	380
9.12.8	BURSTSIZE	380
9.12.9	TXFILLTUNING	381
9.12.10	CONFIGFLAG	381
9.12.11	PORTSCx	382
9.12.11.1	Host Controller	382
9.12.12	USBMODE	387
9.13	Host Data Structures	387
9.13.1	Periodic Frame List	388
9.13.2	Asynchronous List Queue Head Pointer	390
9.13.3	Isochronous (High-Speed) Transfer Descriptor (iTd)	391
9.13.3.1	Next Link Pointer	392
9.13.3.2	iTd Transaction Status and Control List	393
9.13.3.3	iTd Buffer Page Pointer List (Plus)	394
9.13.4	Split Transaction Isochronous Transfer Descriptor (siTd)	395
9.13.4.1	Next Link Pointer	395
9.13.4.2	siTd Endpoint Capabilities/Characteristics	396
9.13.4.3	siTd Transfer State	397
9.13.4.4	siTd Buffer Pointer List (Plus)	398
9.13.4.5	siTd Back Link Pointer	398
9.13.5	Queue Element Transfer Descriptor (qTd)	399
9.13.5.1	Next qTd Pointer	400
9.13.5.2	Alternate Next qTd Pointer	400
9.13.5.3	qTd Token	400
9.13.5.4	qTd Buffer Page Pointer List	403
9.13.6	Queue Head	404
9.13.6.1	Queue Head Horizontal Link Pointer	404
9.13.6.2	Endpoint Capabilities/Characteristics	405
9.13.6.3	Transfer Overlay	407
9.13.7	Periodic Frame Span Traversal Node (FSTN)	408
9.13.7.1	FSTN Normal Path Pointer	408
9.13.7.2	FSTN Back Path Link Pointer	409
9.14	Host Operational Model	409
9.14.1	Host Controller Initialization	410
9.14.2	Port Routing and Control	411
9.14.2.1	Port Routing Control via EHCI <i>Configured (CF) Bit</i>	412



9.14.2.2	Port Routing Control via PortOwner and Disconnect Event	413
9.14.2.3	Example Port Routing State Machine	414
9.14.2.4	Port Power	415
9.14.2.5	Port Reporting Over-Current	415
9.14.3	Suspend/Resume	416
9.14.3.1	Port Suspend/Resume	416
9.14.4	Schedule Traversal Rules	418
9.14.4.1	Example: Preserving Micro-Frame Integrity	420
9.14.5	Periodic Schedule Frame Boundaries versus Bus Frame Boundaries	423
9.14.6	Periodic Schedule	425
9.14.7	Managing Isochronous Transfers Using iTDs	426
9.14.7.1	Host Controller Operational Model for iTDs	426
9.14.7.2	Software Operational Model for iTDs	428
9.14.8	Asynchronous Schedule	430
9.14.8.1	Adding Queue Heads to Asynchronous Schedule	431
9.14.8.2	Removing Queue Heads from Asynchronous Schedule	432
9.14.8.3	Empty Asynchronous Schedule Detection	435
9.14.8.4	Restarting Asynchronous Schedule Before EOF	435
9.14.8.5	Asynchronous Schedule Traversal: Start Event	438
9.14.8.6	Reclamation Status Bit (USBSTS Register)	438
9.14.9	Operational Model for Nak Counter	439
9.14.9.1	Nak Count Reload Control	440
9.14.10	Managing Control/Bulk/Interrupt Transfers via Queue Heads	441
9.14.10.1	Fetch Queue Head	443
9.14.10.2	Advance Queue	443
9.14.10.3	Execute Transaction	444
9.14.10.4	Write Back qTD	449
9.14.10.5	Follow Queue Head Horizontal Pointer	449
9.14.10.6	Buffer Pointer List Use for Data Streaming with qTDs	449
9.14.10.7	Adding Interrupt Queue Heads to the Periodic Schedule	451
9.14.10.8	Managing Transfer Complete Interrupts from Queue Heads	451
9.14.11	Ping Control	452
9.14.12	Split Transactions	453
9.14.12.1	Split Transactions for Asynchronous Transfers	453
9.14.12.2	Split Transaction Interrupt	455
9.14.12.3	Split Transaction Isochronous	468
9.14.13	Host Controller Pause	481
9.14.14	Port Test Modes	481
9.14.15	Interrupts	482
9.14.15.1	Transfer/Transaction Based Interrupts	483
9.14.15.2	Host Controller Event Interrupts	485
9.15	EHCI Deviation	486
9.15.1	Embedded Transaction Translator Function	486
9.15.1.1	Capability Registers	487
9.15.1.2	Operational Registers	487
9.15.1.3	Discovery	487
9.15.1.4	Data Structures	488
9.15.1.5	Operational Model	488
9.15.2	Device Operation	490
9.15.2.1	USBMODE Register	490
9.15.2.2	EHCI Reserved Fields	490
9.15.2.3	SOF Interrupt	491
9.15.3	Embedded Design Interface	491
9.15.3.1	Frame Adjust Register	491
9.15.4	Miscellaneous Variations from EHCI	491
9.15.4.1	Programmable Physical Interface Behavior	491
9.15.4.2	Discovery	491



9.15.4.3	Port Test Mode	492
9.16	Compatibility	493
9.17	Power-Management Requirements	493
9.17.1	USB Power States	493
9.17.2	Host Power States	493
9.18	Error/Abnormal Conditions	493
10.0	PCI Controller	495
10.1	Introduction	495
10.2	Overview	495
10.2.1	List of Features	500
10.2.2	PCI Controller Configured as Host	501
10.2.2.1	Example: Generating a PCI Configuration Write and Read	503
10.2.3	PCI Controller Configured as Option	504
10.2.4	Initializing PCI Controller Configuration and Status Registers for Data Transactions	505
10.2.4.1	Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register	507
10.2.4.2	Example: PCI Memory Base Address Register and South-AHB Translation	508
10.2.5	Initializing the PCI Controller Configuration Registers	509
10.2.6	PCI Controller South AHB Transactions	512
10.2.7	PCI Controller Functioning as Bus Initiator	512
10.2.7.1	Initiated Type-0 Read Transaction	513
10.2.7.2	Initiated Type-0 Write Transaction	513
10.2.7.3	Initiated Type-1 Read Transaction	514
10.2.7.4	Initiated Type-1 Write Transaction	515
10.2.7.5	Initiated Memory Read Transaction	516
10.2.7.6	Initiated Memory Write Transaction	517
10.2.7.7	Initiated I/O Read Transaction	517
10.2.7.8	Initiated I/O Write Transaction	518
10.2.7.9	Initiated Burst Memory Read Transaction	519
10.2.7.10	Initiated Burst Memory Write Transaction	520
10.2.8	PCI Controller Functioning as Bus Target	521
10.2.9	PCI Controller Door Bell Register	521
10.3	Functional Description	522
10.3.1	PCI Byte-Enable Generation	522
10.3.2	PCI Core	522
10.3.2.1	PCI Target Interface	523
10.3.2.2	PCI Initiator Interface	524
10.3.2.3	PCI Host Functions	526
10.3.2.4	PCI Controller Clock and Reset Generation	528
10.3.2.5	PCI Configuration Register Access	528
10.3.2.6	PCI Pad Drive Strength Compensation Support	529
10.3.2.7	AHB Master Interface	530
10.3.2.8	AHB Master Writes	531
10.3.2.9	AHB Master Reads	532
10.3.2.10	AHB Slave Interface	532
10.3.2.11	PCI Byte Enable Generation	535
10.3.3	PCI Controller DMA	536
10.3.3.1	AHB-to-PCI DMA Channel Operation	540
10.3.3.2	PCI-to-AHB DMA Channel Operation	540
10.3.4	Data Byte Alignment and Addressing — PCI Endianness	541
10.3.5	PCI Controller Interrupts	547
10.3.5.1	PCI Interrupt Generation	547
10.3.5.2	Internal Interrupt Generation	548
10.4	PCI RCOMP Circuitry	549



10.5	Register Descriptions.....	549
10.5.1	PCI Configuration Registers.....	549
10.5.2	PCI Configuration Register Descriptions	550
10.5.2.1	Device ID/Vendor ID Register	550
10.5.2.2	Status Register/Control Register.....	551
10.5.2.3	Class Code/Revision ID Register	552
10.5.2.4	BIST/Header Type/Latency Timer/Cache Line Register.....	553
10.5.2.5	Base Address 0 Register.....	553
10.5.2.6	Base Address 1 Register.....	554
10.5.2.7	Base Address 2 Register.....	554
10.5.2.8	Base Address 3 Register.....	555
10.5.2.9	Base Address 4 Register.....	555
10.5.2.10	Base Address 5 Register.....	556
10.5.2.11	Subsystem ID/Subsystem Vendor ID Register	556
10.5.2.12	Max_Lat, Min_gnt, Interrupt Pin, and Interrupt Line Register.....	557
10.5.2.13	Retry Timeout/trdy Timeout Register	557
10.5.3	PCI Controller Configuration and Status Registers (CSRs).....	558
10.5.3.1	PCI Controller Non-Prefetch Address Register	559
10.5.3.2	PCI Controller Non-Prefetch Command/Byte Enables Register.....	559
10.5.3.3	PCI Controller Non-Prefetch Write Data Register	560
10.5.3.4	PCI Controller Non-Prefetch Read Data Register	560
10.5.3.5	PCI Controller Configuration Port Address/Command/Byte Enables Register	561
10.5.3.6	PCI Controller Configuration Port Write Data Register	562
10.5.3.7	PCI Controller Configuration Port Read Data Register	562
10.5.3.8	PCI Controller Control and Status Register	563
10.5.3.9	PCI Controller Interrupt Status Register	564
10.5.3.10	PCI Controller Interrupt Enable Register	565
10.5.3.11	DMA Control Register.....	565
10.5.3.12	AHB Memory Base Address Register.....	566
10.5.3.13	AHB I/O Base Address Register	567
10.5.3.14	PCI Memory Base Address Register.....	567
10.5.3.15	AHB Doorbell Register	568
10.5.3.16	PCI Doorbell Register	569
10.5.3.17	AHB-to-PCI DMA AHB Address Register 0	569
10.5.3.18	AHB-to-PCI DMA PCI Address Register 0.....	570
10.5.3.19	AHB-to-PCI DMA Length Register 0.....	570
10.5.3.20	AHB-to-PCI DMA AHB Address Register 1	571
10.5.3.21	AHB-to-PCI DMA PCI Address Register 1.....	571
10.5.3.22	AHB-to-PCI DMA Length Register 1.....	572
10.5.3.23	PCI-to-AHB DMA AHB Address Register 0	572
10.5.3.24	PCI-to-AHB DMA PCI Address Register 0.....	573
10.5.3.25	PCI-to-AHB DMA Length Register 0.....	573
10.5.3.26	PCI-to-AHB DMA AHB Address Register 1	574
10.5.3.27	PCI-to-AHB DMA PCI Address Register 1.....	574
10.5.3.28	PCI-to-AHB DMA Length Register 1.....	575
10.6	Error/Abnormal Conditions.....	575
10.6.1	Error Handling as a PCI Target	575
10.6.2	Error Handling as a PCI Initiator During PCI Direct Access from the AHB Bus	577
10.6.3	Error Handling as a PCI Initiator During Non-Prefetch Operations	578
10.6.4	Error Handling During PCI-to-AHB DMA Channel Operations	578
10.6.5	Error Handling During AHB-to-PCI DMA Channel Operations	579
11.0	Memory Controller.....	581
11.1	Overview	581
11.2	Theory of Operation	582
11.2.1	Functional Blocks	582
11.2.1.1	Transaction Ports	584



11.2.1.2	Address Decode Blocks	585
11.2.1.3	Memory Transaction Queues	586
11.2.1.4	Configuration Registers.....	586
11.2.1.5	Refresh Counter	586
11.2.1.6	DDR1 SDRAM Control Block	587
11.2.1.7	DDR1 SDRAM RCOMP Block	587
11.2.2	DDR1 SDRAM Memory Support	587
11.2.2.1	DDR1 SDRAM Interface	588
11.2.2.2	DDR1 SDRAM Bank Sizes and Configurations	590
11.2.2.3	MTCTR Register Setup	593
11.2.2.4	DDR1 SDRAM Addressing	593
11.2.2.5	32-Bit Data Bus Width	594
11.2.2.6	Page Hit/Miss Determination	595
11.2.2.7	DDR1 SDRAM Commands	598
11.2.2.8	DDR1 SDRAM Initialization.....	598
11.2.2.9	DDR1 SDRAM Mode Programming.....	602
11.2.2.10	DDR1 SDRAM Read Cycle	606
11.2.2.11	DDR1 SDRAM Write Cycle	609
11.2.2.12	DDR1 SDRAM Refresh Cycle.....	612
11.2.3	Error Correction and Detection	614
11.2.3.1	ECC Generation	615
11.2.3.2	ECC Generation for Partial Writes	617
11.2.3.3	ECC Checking	620
11.2.3.4	Scrubbing	624
11.2.3.5	ECC Disabled.....	625
11.2.3.6	ECC Testing	625
11.2.4	Overlapping Memory Regions.....	626
11.2.5	DDR1 SDRAM Clocking	626
11.2.6	Performance Monitoring	626
11.3	Power Failure Mode	627
11.4	Interrupts/Error Conditions.....	627
11.4.1	Single-Bit Error Detection	628
11.4.2	Multi-Bit Error Detection.....	629
11.5	Reset Conditions	629
11.6	Register Definitions	630
11.6.1	DDR1 SDRAM Initialization Register SDIR.....	632
11.6.2	DDR1 SDRAM Control Register 0 SDCR0	633
11.6.3	DDR1 SDRAM Control Register 1 SDCR1	635
11.6.4	DDR1 SDRAM Base Register SDBR	637
11.6.5	DDR1 SDRAM Boundary Register 0 SBR0	638
11.6.6	DDR1 SDRAM Boundary Register 1 SBR1	639
11.6.7	ECC Control Register ECCR	640
11.6.8	ECC Log Registers ELOG0, ELOG1.....	641
11.6.9	ECC Address Registers ECAR0, ECAR1	642
11.6.10	ECC Test Register ECTST	643
11.6.11	Memory Controller Interrupt Status Register MCISR	644
11.6.12	MCU Port Transaction Count Register MPTCR.....	645
11.6.13	MCU Preemption Control Register MPCR	645
11.6.14	Refresh Frequency Register RFR	646
11.6.15	SDRAM Page Registers SDPR0-7.....	647
12.0	Expansion Bus Controller	649
12.1	Overview	649
12.2	Feature List	649
12.3	Block Diagram	650
12.4	Theory of Operation.....	650
12.4.1	Outbound Transfers	650



12.4.1.1	Expansion Bus Address Space	653
12.4.1.2	Chip Select Address Allocation.....	653
12.4.1.3	Address and Data Byte Steering	655
12.4.1.4	Expansion Bus Interface Configuration	658
12.4.1.5	Using I/O Wait	661
12.4.1.6	Parity	663
12.4.1.7	Special Design Knowledge for Using HPI mode	664
12.4.1.8	Expansion Bus Outbound Timing Diagrams	665
12.4.2	Inbound Transfers	681
12.4.2.1	Parity	686
12.4.3	Arbitration	686
12.4.3.1	Internal Arbitration.....	687
12.4.3.2	External Arbiter.....	688
12.4.4	Multiple Processors Connected by the Expansion Bus	688
12.4.5	Expansion Bus Inbound Timing Diagrams.....	690
12.4.5.1	Back-to-Back 1-Word Inbound Write with EX_SLAVE_CS_N Deasserted.....	690
12.4.5.2	Back-to-Back 1-Word Writes without Deasserting EX_SLAVE_CS_N.....	690
12.4.5.3	Eight-Word Inbound Write	691
12.4.5.4	Eight-Word Inbound Write with Master Wait States	692
12.4.5.5	Eight-Word Inbound Write with NOPS.....	693
12.4.5.6	Eight-Word Inbound Write with EX_SLAVE_CS_N Deassertion	694
12.4.5.7	Back-to-Back 1-Word Inbound Reads with EX_SLAVE_CS_N.....	695
12.4.5.8	Back-to-Back 1-Word Reads without EX_SLAVE_CS_N Deasserted.....	696
12.4.5.9	Eight-Word Inbound Read	696
12.4.5.10	Eight-Word Inbound Read with Master Wait States	697
12.4.5.11	Eight-Word Inbound Read with Deassertion of EX_SLAVE_CS_N	698
12.4.6	Expansion Bus Arbiter Timing Diagrams.....	698
12.4.6.1	Arbitration When GrantRemove Bit In EXP_MST_CONTROL is Set.....	698
12.4.6.2	Arbitration When GrantRemove Bit in EXP_MST_CONTROL is Clear	699
12.4.7	External Expansion Bus Timing Diagram	700
12.4.7.1	External Arbiter Timing Diagram.....	700
12.4.8	Configuration Straps	700
12.4.8.1	Sampling EX_ADDR During Reset	701
12.4.8.2	Expansion Bus Controller Operation	701
12.5	Detailed Register Descriptions	701
12.5.1	Timing and Control Registers for Chip Select 0	703
12.5.2	Timing and Control Registers for Chip Select 1	703
12.5.3	Timing and Control Registers for Chip Select 2	703
12.5.4	Timing and Control Registers for Chip Select 3	704
12.5.5	Timing and Control Registers for Chip Select 4	704
12.5.6	Timing and Control Registers for Chip Select 5	704
12.5.7	Timing and Control Registers for Chip Select 6	705
12.5.8	Timing and Control Registers for Chip Select 7	705
12.5.9	Configuration Register 0	706
12.5.10	Configuration Register 1	709
12.5.11	EXP_UNIT_FUSE_RESET	713
12.5.12	EXP_SMIIDLL	716
12.5.13	EXP_MST_CONTROL.....	716
12.5.14	EXP_INBOUND_ADDR.....	717
12.5.15	EXP_LOCK0	719
12.5.16	EXP_LOCK1	720
12.5.17	EXP_PARITY_STATUS	721



12.5.18	EXP_SYNCINTEL_COUNT	722
13.0	HSS Coprocessor	723
13.1	Overview	723
13.1.1	High-Speed Serial Interface Receive Operation	724
13.1.2	High-Speed Serial Interface Transmit Operation	725
13.2	Feature List	725
13.3	Theory of Operation	726
13.3.1	FIFOs and Lookup Tables	727
13.3.1.1	FIFOs	727
13.3.1.2	Lookup Tables	727
13.3.2	Endianness	730
13.3.3	Programmable Frame Pulse Offset and Frame Synchronization	730
13.3.4	Underflow/Overflow/Unexpected Frame Pulse	732
13.3.5	56K Mode	733
13.3.6	Frameless Data Protocol Support	733
13.3.7	Loopback	733
13.4	HSS Registers and Clock Configuration	734
13.4.1	HSS Clock and Jitter	734
13.4.2	Overview of HSS Clock Configuration	735
13.5	HSS Supported Framing Protocols	736
13.5.1	T1	736
13.5.2	E1	738
13.5.3	GCI	740
13.5.3.1	Line-Card Mode	740
13.5.3.2	Termination Mode	741
13.5.4	MVIP	742
13.5.4.1	2.048-Mbps Backplane	743
13.5.4.2	4.096-Mbps Backplane	744
13.5.4.3	8.192-Mbps Backplane	746
14.0	Universal Asynchronous Receiver-Transmitter (UART)	749
14.1	Overview	749
14.2	Feature List	750
14.3	Block Diagram	751
14.4	Theory of Operation	752
14.4.1	Setting the Baud Rate	753
14.4.2	Setting Data Bits/Stop Bits/Parity	753
14.4.3	Using the Modem Control Signals	756
14.4.4	UART Interrupts	757
14.4.5	Transmitting and Receiving UART Data	760
14.5	Register Descriptions	761
14.5.1	Receive Buffer Register	762
14.5.2	Transmit Holding Register	763
14.5.3	Divisor Latch Low Register	763
14.5.4	Divisor Latch High Register	764
14.5.5	Interrupt Enable Register	764
14.5.6	Interrupt Identification Register	765
14.5.7	FIFO Control Register	767
14.5.8	Line Control Register	768
14.5.9	Modem Control Register	770
14.5.10	Line Status Register	771
14.5.11	Modem Status Register	772
14.5.12	Scratch-Pad Register	773
14.5.13	Infrared Selection Register	774



15.0 GPIO Controller	776
15.1 Overview	776
15.2 Feature List	776
15.3 Block Diagram	776
15.4 Theory of Operation	777
15.4.1 Input Meta-Stability Protection, Edge Detect Logic, Pulse Discrimination	778
15.4.2 Clock Generation	778
15.4.3 APB Interface	779
15.5 Detailed Register Descriptions	779
15.5.1 GPIO Output Register	779
15.5.2 GPIO Output Enable Register	780
15.5.3 GPIO Input Register	781
15.5.4 GPIO Interrupt Status Register	781
15.5.5 GPIO Interrupt Type Register 1	782
15.5.6 GPIO Interrupt Type Register 2	783
15.5.7 GPIO Clock Register	784
16.0 Performance Monitoring Unit (PMU)	787
16.1 Overview	787
16.2 Feature List	787
16.3 Functional Description	787
16.3.1 Programmable Event Counters	788
16.3.2 Occurrence Events	788
16.3.3 Duration Events	789
16.3.4 Performance Monitoring	791
16.3.4.1 Halt: Performance Monitoring Disabled	791
16.3.4.2 Cycle Count	791
16.3.4.3 MCU: DRAM Transactions	791
16.3.4.4 Events	792
16.4 Previous Master and Slave	792
16.5 Miscellaneous	792
16.5.1 Interrupts	792
16.5.2 Reset Conditions	793
16.6 Detailed Register Descriptions	793
16.6.1 Event Select Registers	794
16.6.2 PMU Status Register	795
16.6.3 PMU Mode Register	795
16.6.4 Programmable Event Counters	796
16.6.5 Previous Master/Slave Register	797
16.7 Event Mapping	798
17.0 Interrupt Controller	803
17.1 Overview	803
17.2 Feature List	805
17.3 Block Diagram	806
17.4 Theory of Operation	806
17.4.1 Interrupt Priority	807
17.4.2 Assigning FIQ or IRQ Interrupts	808
17.4.3 Enabling and Disabling Interrupts	809
17.4.4 Reading Interrupt Status	809
17.5 Error Enable Register	811
17.6 Interrupt Controller Register Descriptions	811
17.6.1 Interrupt Status Register	812
17.6.2 Interrupt Enable Register	813
17.6.3 Interrupt Select Register	813
17.6.4 IRQ Status Register	814



17.6.5	FIQ Status Register	814
17.6.6	Interrupt Priority Register.....	815
17.6.7	IRQ Highest-Priority Register	815
17.6.8	FIQ Highest-Priority Register.....	816
17.6.9	Error High Priority Enable Register	816
18.0	Operating System Timer	817
18.1	Overview	817
18.2	Feature List	817
18.3	Block Diagram	817
18.4	Theory of Operation.....	818
18.4.1	Watchdog Timer Operation	818
18.4.2	Timestamp Timer Operation.....	819
18.4.3	General-Purpose Timers Operation	820
18.4.4	Clock Prescale.....	821
18.5	Detailed Register Descriptions.....	821
18.5.1	Timestamp Timer	822
18.5.2	General-Purpose Timer 0.....	822
18.5.3	General-Purpose Timer 0 Reload	823
18.5.4	General-Purpose Timer 1.....	823
18.5.5	General-Purpose Timer 1 Reload	824
18.5.6	Watchdog Timer.....	824
18.5.7	Watchdog Enable Register	825
18.5.8	Watchdog Key Register	825
18.5.9	Timer Status.....	826
18.5.10	Timestamp Compare Register	826
18.5.11	Timestamp Configuration Register	827
18.5.12	Timestamp Prescale Register	827
18.5.13	General-Purpose Timer 0 Configuration Register	828
18.5.14	General-Purpose Timer 0 Prescale Register	828
18.5.15	General-Purpose Timer 1 Configuration Register	829
18.5.16	General-Purpose Timer 1 Prescale Register	829
19.0	Time Synchronization Hardware Assist (TSYNC)	830
19.1	Overview	830
19.2	Block Diagram	830
19.3	Theory of Operation (Ethernet Interfaces)	831
19.3.1	Priority Message Support.....	832
19.3.2	Sync Message.....	832
19.3.3	Follow_Up Message	832
19.3.4	Delay_Req Message.....	833
19.3.5	Delay_Resp Message	833
19.3.6	IPv6 Compatibility	833
19.3.7	Traffic Analyzer Support.....	833
19.3.8	MII Clocking Methods.....	833
19.3.9	System Time Clock Rate Set by Addend Register	833
19.3.10	MII Message Detection.....	834
19.3.10.1	Sync Message	835
19.3.10.2	Delay_Req Message	835
19.3.10.3	Errors in Messages.....	835
19.4	Theory of Operation (Auxiliary Snapshots)	836
19.4.1	Master Mode Programming Considerations.....	836
19.4.2	Slave Mode Programming Considerations.....	836
19.5	Detailed Register Descriptions.....	836
19.5.1	Register Map	836
19.5.2	Register Descriptions	838



19.5.2.1	Time Sync Control Register.....	839
19.5.2.2	Time Sync Event Register.....	840
19.5.2.3	Addend Register.....	840
19.5.2.4	Accumulator Register.....	841
19.5.2.5	Test Register	842
19.5.2.6	RawSystemTime_Low Register	843
19.5.2.7	RawSystemTime_High Register	843
19.5.2.8	SystemTime_Low Register.....	844
19.5.2.9	SystemTime_High Register.....	844
19.5.2.10	TargetTime_Low Register	845
19.5.2.11	TargetTime_High Register	845
19.5.2.12	Auxiliary Slave Mode Snapshot Low Register – ASMS_Low	846
19.5.2.13	Auxiliary Slave Mode Snapshot High Register – ASMS_High	846
19.5.2.14	Auxiliary Master Mode Snapshot Low Register – AMMS_Low	847
19.5.2.15	Auxiliary Master Mode Snapshot High Register – AMMS_High	847
19.5.2.16	TS_Channel_Control Register (Per Channel).....	848
19.5.2.17	TS_Channel_Event Register (Per Channel)	849
19.5.2.18	XMIT_Snapshot_Low Register (Per Channel)	850
19.5.2.19	XMIT_Snapshot_High Register (Per Channel)	851
19.5.2.20	RECV_Snapshot Low Register (Per Channel)	852
19.5.2.21	RECV_Snapshot High Register (Per Channel).....	853
19.5.2.22	SourceUUID0_Low Register (Per Channel)	854
19.5.2.23	SequenceID/SourceUUID_High Register (Per Channel)	855
20.0	Synchronous Serial Port	857
20.1	SSP Operation.....	857
20.1.1	Processor-Initiated Data Transfer	857
20.2	Data Formats	858
20.2.1	Serial Data Formats for Transfer to/from Peripherals	858
20.2.1.1	SSP Format — Detail	859
20.2.1.2	SPI Format — Detail	859
20.2.1.3	Microwire* Format — Details.....	861
20.2.2	Parallel Data Formats for Buffer Storage	862
20.3	Buffer Operation.....	862
20.4	Baud-Rate Generation	863
20.5	SSP Serial Port Registers	863
20.5.1	SSP Control Register 0 (SSCR0).....	864
20.5.1.1	Data Size Select (DSS)	864
20.5.1.2	Frame Format (FRF)	864
20.5.1.3	External Clock Select (ECS)	864
20.5.1.4	Synchronous Serial Port Enable (SSE)	864
20.5.1.5	Serial Clock Rate (SCR).....	865
20.5.2	SSP Control Register 1 (SSCR1).....	866
20.5.2.1	Receive FIFO Interrupt Enable (RIE)	866
20.5.2.2	Transmit FIFO Interrupt Enable (TIE)	866
20.5.2.3	Loop Back Mode (LBM).....	866
20.5.2.4	Serial Clock Polarity (SPO).....	867
20.5.2.5	Serial Clock Phase (SPH)	867
20.5.2.6	National Microwire* Data Size (MWDS)	868
20.5.2.7	Transmit FIFO Interrupt Threshold (TFT)	868
20.5.2.8	Receive FIFO Interrupt Threshold (RFT).....	868
20.5.2.9	Enable FIFO Write/Read Function (EFWR)	868
20.5.2.10	Select FIFO for Enable FIFO Write/Read (STRF)	868
20.5.3	SSP Status Register	869
20.5.3.1	Transmit FIFO Not Full Flag (TNF) (Read-Only, Non-Interruptible) ..	870
20.5.3.2	Receive FIFO Not Empty Flag (RNE) (Read-Only, Non-Interruptible)	870
20.5.3.3	SSP Busy Flag (BSY) (Read-Only, Non-Interruptible).....	870



20.5.3.4	Transmit FIFO Service Request Flag (TFS) (Read-Only, Maskable Interrupt)	870
20.5.3.5	Receive FIFO Service Request Flag (RFS) (Read-Only, Maskable Interrupt)	870
20.5.3.6	Receiver Overrun Status (ROR)	870
20.5.3.7	Transmit FIFO Level (TFL)	871
20.5.3.8	Receive FIFO Level (RFL)	871
20.5.4	SSP Interrupt Test Register (SSITR)	872
20.5.5	SSP Data Register (SSDR)	872
21.0	I2C Bus Interface Unit	875
21.1	Overview	875
21.2	Feature List	875
21.3	Block Diagram	875
21.4	Theory of Operation	876
21.4.1	Operational Blocks	877
21.4.2	I ² C Bus Interface Modes	878
21.4.3	Start and Stop Bus States	879
21.4.3.1	START Condition	880
21.4.3.2	No START or STOP Condition	880
21.4.3.3	STOP Condition	881
21.5	I ² C Bus Operation	881
21.5.1	Serial Clock Line (SCL) Generation	881
21.5.2	Data and Addressing Management	882
21.5.2.1	Addressing a Slave Device	882
21.5.3	I ² C Acknowledge	883
21.5.4	Arbitration	884
21.5.4.1	SCL Arbitration	884
21.5.4.2	SDA Arbitration	885
21.5.5	Master Operations	886
21.5.6	Slave Operations	889
21.5.7	General Call Address	891
21.6	Slave Mode Programming Examples	892
21.6.1	Initialize Unit	892
21.6.2	Write n Bytes as a Slave	892
21.6.3	Read n Bytes as a Slave	893
21.7	Master Programming Examples	894
21.7.1	Initialize Unit	894
21.7.2	Write 1 Byte as a Master	894
21.7.3	Read 1 Byte as a Master	894
21.7.4	Write 2 Bytes and Repeated Start Read 1 Byte as a Master	895
21.7.5	Read 2 Bytes as a Master — Send STOP Using the Abort	895
21.8	Glitch Suppression Logic	896
21.9	Reset Conditions	896
21.10	Register Definitions	896
21.10.1	I ² C Control Register - ICR	897
21.10.2	I ² C Status Register - ISR	899
21.10.3	I ² C Slave Address Register - ISAR	901
21.10.4	I ² C Data Buffer Register - IDBR	902
21.10.5	I ² C Bus Monitor Register - IBMR	902
22.0	Public Key Exchange Crypto Engine	905
23.0	AHB-PKE Bridge	907
23.1	Overview	907
23.2	Feature List	907
23.3	Block Diagram	908



23.4	Theory of Operation	908
23.4.1	Peripheral Information.....	908
24.0	Random Number Generator	911
24.1	Theory of Operation	911
24.2	Detailed Register Descriptions	911
24.2.1	Registers	912
24.2.1.1	Random Number FIFO.....	912
24.3	Power-Management Requirements	912
24.4	Reset	912
24.5	Error/Abnormal Conditions	912
25.0	Exponentiation Acceleration Unit	913
25.1	Overview	913
25.2	Feature List	913
25.3	Block Diagram.....	914
25.3.1	Operand Restrictions.....	914
25.3.2	EAU RAM Writes	915
25.3.3	EAU RAM Reads.....	915
25.4	Detailed Register Descriptions	915
25.4.1	EAU Command Register	915
25.4.2	EAU Status Register	917
25.4.3	EAU Count Register.....	917
25.4.4	EAU Interrupt Register	918
25.4.5	EAU RAM Registers	918
25.5	Performance Requirements	919
26.0	Hashing Unit (SHA)	921
26.1	Overview.....	921
26.2	Feature List	921
26.3	Block Diagram.....	921
26.4	Detailed Register Descriptions	921
26.4.1	Hash Configuration Register	922
26.4.2	Hash Do Register	922
26.4.3	Hash Interrupt Register	923
26.4.4	Hash Chain Register	923
26.4.5	Hash Data FIFO	924
26.5	Compatibility.....	924
27.0	AHB Queue Manager (AQM).....	926
27.1	Overview.....	926
27.2	Feature List	926
27.3	Block Diagram.....	927
27.4	AHB Interface	928
27.4.1	Queue Control.....	929
27.4.2	Queue Status	933
27.4.2.1	Status Update.....	933
27.4.2.2	Status Interrupts.....	934
27.4.3	AQM SRAM	935
27.4.4	Data Validity	936
27.4.5	Burst Operations to Queues	937
27.5	Detailed Register Descriptions	937
27.6	Register Descriptions.....	939
27.6.1	Queue Access Word Registers 0 - 63	939
27.6.2	Queues 0-31 Status Register 0 - 3	939
27.6.3	Underflow/Overflow Status Register 0 - 1.....	940
27.6.4	Queues 32-63 Empty Status Register	940



27.6.5	Queues 32-63 Nearly Empty Status Register	941
27.6.6	Queues 32-63 Nearly Full Status Register	942
27.6.7	Queues 32-63 Full Status Register	942
27.6.8	Interrupt 0 Status Flag Source Select Register 0 – 3	942
27.6.9	Queue Interrupt Enable Register 0 – 1	944
27.6.10	Queue Interrupt Register 0 – 1	944
27.6.11	Queue Configuration Words 0 - 63	945
27.6.12	Queue 32 to 63 Event 'A' Enable Register	946
27.6.13	Queue 32 to 63 Event 'B' Enable Register	946
27.6.14	Queue 32 to 63 Event 'C' Enable Register	946
27.6.15	Event Source Select	947
27.6.16	Queue 0 to 31 Status Selection Map Register	948
27.6.17	Queue SRAM Error Data Register	949
27.6.18	Queue SRAM Error Address/Control Register	949
27.7	Error/Abnormal Conditions	949
28.0	Error Handling	951
28.1	Errors	951
28.1.1	Sources of AHB Bus Errors	951
28.1.1.1	Accesses to Reserved or Unimplemented Addresses	951
28.1.1.2	Illegal-Access Types	951
28.1.1.3	Expansion Bus Parity Error	951
28.1.1.4	AQM-Parity Error	951
28.1.1.5	Memory Controller Unit (MCU), Multiple-Bit, ECC Error	951
28.2	Responses to Errors	952
28.2.1	PCI Responses to Errors	952
28.2.2	NPE Responses to Errors	952
28.2.3	Expansion Bus Controller Response to Errors	957
28.2.4	Intel XScale® Processor Response to Errors	957
28.2.5	AHB-AHB Bridge Response to Errors	957
28.3	Multiple Error Conditions	957

Figures

1	Intel® IXP465 Network Processor Block Diagram	45
2	Intel® IXP460 Network Processor Block Diagram	46
3	Intel® IXP455 Network Processor Block Diagram	47
4	Intel XScale® Processor Block Diagram	62
5	Example of Locked Entries in TLB	77
6	Instruction Cache Organization	78
7	Locked Line Effect on Round-Robin Replacement	82
8	BTB Entry	84
9	Branch History	84
10	Data Cache Organization	86
11	Mini-Data Cache Organization	87
12	Locked Line Effect on Round-Robin Replacement	95
13	SELDCSR Hardware	125
14	SELDCSR Data Register	126
15	DBGTX Hardware	128
16	DBGRX Hardware	129
17	Rx Write Logic	130
18	DBGRX Data Register	131
19	Message Byte Formats	134
20	Indirect Branch Entry Address Byte Organization	137
21	High-Level View of Trace Buffer	137



22	LDIC JTAG Data Register Hardware.....	140
23	Format of LDIC Cache Functions.....	142
24	Code Download During a Cold Reset For Debug.....	144
25	Code Download During a Warm Reset For Debug.....	146
26	Downloading Code in IC During Program Execution.....	147
27	RISC Super-Pipeline.....	191
28	Multiple Ethernet PHYS Connected to Processor.....	229
29	Ethernet Coprocessor Interface.....	230
30	MDIO Write.....	232
31	MDIO Read.....	233
32	UTOPIA Level 2 Coprocessor.....	269
33	UTOPIA Level 2 MPHY Transmit Polling.....	271
34	UTOPIA Level 2 MPHY Receive Polling.....	274
35	NRZI Bit Encoding Example.....	282
36	Example USB 2.0 System Configuration.....	354
37	Top-Level Block Diagram.....	355
38	Periodic Schedule Organization.....	356
39	Asynchronous Schedule Organization.....	357
40	Block Diagram.....	358
41	Microprocessor Interface Block Diagram.....	359
42	DMA Engine Block Diagram.....	360
43	Protocol Engine Block Diagram.....	361
44	Port Controller Block Diagram.....	362
45	Periodic Schedule Organization.....	390
46	Asynchronous Schedule Organization.....	391
47	Isochronous Transaction Descriptor (iTD).....	392
48	Split-transaction Isochronous Transaction Descriptor (siTD).....	395
49	Queue Element Transfer Descriptor Block Diagram.....	399
50	Queue Head Structure Layout.....	404
51	Frame Span Traversal Node Structure Layout.....	408
52	Example USB 2.0 Host Controller Port Routing Block Diagram.....	411
53	Port Owner Hand-Off State Machine.....	414
54	Derivation of Pointer into Frame List Array.....	419
55	General Format of Asynchronous Schedule List.....	419
56	Best Fit Approximation.....	421
57	Frame Boundary Relationship between HS bus and FS/LS Bus.....	423
58	Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries.....	424
59	Example Periodic Schedule.....	426
60	Example Association of iTDs to Client Request Buffer.....	429
61	Generic Queue Head Unlink Scenario.....	434
62	Asynchronous Schedule List with Annotation to Mark Head of List.....	435
63	Example State Machine for Managing Asynchronous Schedule Traversal.....	436
64	Example HC State Machine for Controlling Nak Counter Reloads.....	440
65	Host Controller Queue Head Traversal State Machine.....	442
66	Example Mapping of qTD Buffer Pointers to Buffer Pages.....	450
67	Host Controller Asynchronous Schedule Split-Transaction State Machine.....	454
68	Split Transaction, Interrupt Scheduling Boundary Conditions.....	456
69	General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading.....	457
70	Example Host Controller Traversal of Recovery Path via FSTNs.....	459
71	Split Transaction State Machine for Interrupt.....	463
72	Split Transaction, Isochronous Scheduling Boundary Conditions.....	469
73	siTD Scheduling Boundary Examples.....	471
74	Split Transaction State Machine for Isochronous.....	474
75	PCI Bus Configured as a Host.....	496
76	PCI Bus Configured as an Option.....	496



77	PCI Controller Block Diagram	497
78	Type 0 Configuration Address Phase	502
79	Type 1 Configuration Address Phase	502
80	Initiated PCI Type-0 Configuration Read Cycle	513
81	Initiated PCI Type-0 Configuration Write Cycle	514
82	Initiated PCI Type-1 Configuration Read Cycle	515
83	Initiated PCI Type-1 Configuration Write Cycle	516
84	Initiated PCI Memory Read Cycle	516
85	Initiated PCI Memory Write Cycle	517
86	Initiated PCI I/O Read Cycle	518
87	Initiated PCI I/O Write Cycle	519
88	Initiated PCI Burst Memory Read Cycle	520
89	Initiated PCI Burst Memory Write Cycle	521
90	PCI Controller Arbiter Configuration	527
91	PCI-to-AHB Address Translation	531
92	AHB-to-PCI Address Translation – Memory Cycles	533
93	AHB-to-PCI DMA-Transfer Byte Lane Swapping	538
94	PCI-to-AHB DMA-Transfer Byte Lane Swapping	538
95	Byte Lane Routing During PCI Target Accesses of the AHB Bus – Big-Endian AHB Bus	542
96	Byte Lane Routing During PCI Target Accesses of the AHB Bus – Little-Endian AHB Bus...	543
97	Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Big-Endian AHB Bus	544
98	Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Little-Endian AHB Bus	545
99	Byte Lane Routing During DMA Transfers	546
100	Byte Lane Routing During CSR Accesses	547
101	Memory Controller Block Diagram	584
102	Dual-Bank DDRI SDRAM Memory Subsystem	589
103	64-Bit to 32-Bit Addressing	594
104	Page Hit/Miss Logic for 128/256/512/1, 024-Bit Mode	596
105	Logical Memory Image of a DDRI SDRAM Memory Subsystem	597
106	Supported DDRI SDRAM Extended Mode Register Settings	599
107	Supported DDRII SDRAM Extended Mode Register Settings	600
108	Supported DDRI SDRAM Mode Register Settings	600
109	DDRI SDRAM Initialization Sequence (Controlled with Software)	601
110	MCU Active, Precharge, Refresh Command Timing Diagram	603
111	MCU DDR Read Command to Next Command Timing Diagram	604
112	MCU DDR Write Command to Next Command Timing Diagrams	605
113	DDRI SDRAM Pipelined Reads	606
114	DDRI SDRAM Read, 36 Bytes, ECC Enabled, BL=4	607
115	DDRI SDRAM Write, 36 Bytes, ECC Enabled, BL=4	610
116	DDRI SDRAM Pipelined Writes	612
117	Refresh While the Memory Bus is Not Busy	613
118	ECC Write Flow	615
119	IXP45X/IXP46X product line G-Matrix (Generates the ECC)	617
120	Sub 64-bit DDRI SDRAM Write (D ₀)	619
121	ECC Read Data Flow	621
122	Intel® IXP45X and Intel® IXP46X Product Line of Network Processors H-Matrix (Indicates the Single-Bit Error Location)	623
123	Expansion Bus Controller	650
124	Chip Select Address Allocation when there are no 32-MByte Devices Programmed	654
125	Expansion Bus Memory Sizing	654
126	Chip Select Address Allocation when a 32-Mbyte Device is Programmed	655
127	I/O Wait Normal Phase Timing	662
128	I/O Wait Extended Phase Timing	663



129	Expansion Bus Write (Intel, Multiplexed Mode)	666
130	Expansion Bus Read (Intel, Multiplexed Mode)	667
131	Expansion Bus Write (Intel Simplex-Mode, Synchronous Intel)	668
132	Expansion Bus Read (Intel Simplex-Mode)	669
133	Intel Synchronous 8-Word Read	670
134	Intel Synchronous One-Word Read	671
135	Micron* ZBT Write/Read/Write	672
136	Micron* ZBT 8-Word Read	673
137	Micron* ZBT 4-Word Write	674
138	Expansion Bus Write (Motorola*, Multiplexed Mode)	675
139	Expansion Bus Read (Motorola*, Multiplexed Mode)	676
140	Expansion Bus Write (Motorola*, Simplex Mode)	677
141	Expansion Bus Read (Motorola*, Simplex Mode)	678
142	Expansion Bus Write (TI* HPI-8 Mode)	679
143	Expansion Bus Read (TI* HPI-8 Mode)	679
144	Expansion Bus Write (TI* HPI-16, Multiplexed Mode)	680
145	Expansion Bus Read (TI* HPI-16, Multiplexed Mode)	680
146	Expansion Bus Write (TI* HPI-16, Simplex Mode)	681
147	Expansion Bus Read (TI* HPI-16, Simplex Mode)	681
148	Expansion Bus Inbound State Diagram	685
149	Multiple IXP45X/IXP46X network processors Connected Back-to-Back	689
150	Back-to-Back 1-Word Inbound Write with EX_SLAVE_CS_N Deasserted	690
151	Back-to-Back 1-Word Writes without Deasserting EX_SLAVE_CS_N	691
152	Eight-Word Inbound Write	692
153	Eight-Word Inbound Write	693
154	Eight-Word Inbound Write with NOPS	693
155	Eight-Word Inbound Write with EX_SLAVE_CS_N Deassertion	694
156	Back-to-Back 1-Word Inbound Reads with EX_SLAVE_CS_N	695
157	Back-to-Back 1-Word Reads without EX_SLAVE_CS_N Deasserted	696
158	Eight-Word Inbound Read	696
159	Eight-Word Inbound Read with Master Wait States	697
160	Eight-Word Inbound Read with Deassertion of EX_SLAVE_CS_N	698
161	Arbitration When GrantRemove Bit In EXP_MST_CONTROL is Set	698
162	Arbitration When GrantRemove Bit in EXP_MST_CONTROL is Clear	699
163	External Arbiter Timing Diagram	700
164	Sampling EX_ADDR During Reset	701
165	Look-Up Table Organization	728
166	HSS Core RX Buffer Structure (Identical to TX Buffer Structure)	729
167	HSS Endianness Examples	730
168	TX Frame Sync Example (Presuming Zero Offset)	731
169	FRX Frame Sync Example (Presuming Zero Offset)	732
170	T1 TX Frame, HSS Generating Frame Pulse	737
171	T1 TX Frame Using External Frame Pulse	737
172	T1 RX Frame Using External Frame Pulse	738
173	E1 TX Frame, HSS Generating Frame Pulse	739
174	E1 TX Frame, Externally Generated Frame Pulse	739
175	E1 RX Frame, Externally Generated Frame Pulse	740
176	GCI Frames, Internally Generated Frame Pulse (Line-Card Mode)	741
177	GCI Frames, Internally Generated Frame Pulse (Termination Mode)	742
178	MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame	743
179	MVIP, Frame Mapping a T1 Frame to an E1 Frame	744
180	MVIP, Byte Interlacing Two E1 Streams onto a 4.096-Mbps Backplane	745
181	MVIP, Byte Interleaving Two T1 Streams onto a 4.096-Mbps Backplane	746
182	MVIP, Byte Interleaving Four E1 Streams onto a 8.192-Mbps Backplane Bus	746
183	UART Timing Diagram	750



184	UART Block Diagram.....	752
185	GPIO Block Diagram.....	777
186	Interrupt Controller Block Diagram.....	806
187	Operating System Timer Block Diagram.....	818
188	Block Diagram of TSync Circuit.....	831
189	Time Stamp Reference Point.....	832
190	I ² C Bus Interface Unit Block Diagram.....	876
191	I ² C Bus Configuration Example.....	877
192	START and STOP Conditions.....	880
193	START and STOP Condition Events.....	881
194	Data Format of First Byte in Master Transaction.....	883
195	Acknowledge on the I ² C Bus.....	884
196	Clock Synchronization During the Arbitration Procedure.....	885
197	Arbitration Procedure of Two Masters.....	885
198	Master-Receiver Read from Slave-Transmitter.....	888
199	Master-Receiver Read from Slave-Transmitter / Repeated Start / Master-Transmitter Write to Slave-Receiver.....	889
200	A Complete Data Transfer.....	889
201	Master-Transmitter Write to Slave-Receiver.....	891
202	Master-Receiver Read to Slave-Transmitter.....	891
203	Master-Receiver Read to Slave-Transmitter, Repeated START, Master-Transmitter Write to Slave-Receiver.....	891
204	General Call Address.....	892
205	AHB-PKE Bridge Block Diagram.....	908
206	Exponentiation Acceleration Unit: Block Diagram.....	914
207	AHB Queue Manager.....	927
208	Representative Logical Diagram of a Queue.....	931
209	NPE Error Handling Illustration.....	953

Tables

1	List of Acronyms.....	39
2	Register Legend.....	43
3	Supported DDRI Memory Configurations.....	54
4	GPIO Alternate Function Table.....	57
5	Data Cache and Buffer Behavior When X = 0.....	71
6	Data Cache and Buffer Behavior When X = 1.....	72
7	Memory Operations that Impose a Fence.....	72
8	Valid MMU & Data/Mini-Data Cache Combinations.....	73
9	MRC/MCR Format.....	97
10	LDC/STC Format when Accessing CP14.....	97
11	CP15 Registers.....	98
12	ID Register.....	99
13	Cache Type Register.....	99
14	Intel® StrongARM® Control Register.....	100
15	Auxiliary Control Register.....	101
16	Translation Table Base Register.....	102
17	Domain Access Control Register.....	102
18	Fault Status Register.....	103
19	Fault Address Register.....	103
20	Cache Functions.....	104
21	TLB Functions.....	105
22	Cache Lock-Down Functions.....	105
23	Data Cache Lock Register.....	105
24	TLB Lockdown Functions.....	106



25	Accessing Process ID	106
26	Process ID Register.....	106
27	Accessing the Debug Registers	107
28	Coprocessor Access Register	108
29	CP14 Registers.....	109
30	Accessing the Performance Monitoring Registers	109
31	PWRMODE Register.....	110
32	Clock and Power Management	110
33	CCLKCFG Register	110
34	Accessing the Debug Registers	110
35	Debug Control and Status Register (DCSR)	113
36	Event Priority	115
37	Instruction Breakpoint Address and Control Register (IBCRx)	118
38	Data Breakpoint Register (DBRx)	118
39	Data Breakpoint Controls Register (DBCON)	119
40	TX RX Control Register (TXRXCTRL)	121
41	Normal RX Handshaking	121
42	High-Speed Download Handshaking States	122
43	TX Handshaking	123
44	TXRXCTRL Mnemonic Extensions	123
45	TX Register.....	124
46	RX Register	124
47	DEBUG Data Register Reset Values	132
48	CP 14 Trace Buffer Register Summary	133
49	Checkpoint Register (CHKPTx).....	133
50	TBREG Format	134
51	Message Byte Formats	135
52	LDIC Cache Functions	141
53	Debug-Handler Code to Implement Synchronization During Dynamic Code Download	149
54	Debug Handler Code: Download Bit and Overflow Flag.....	155
55	Performance Monitoring Registers.....	157
56	Register Legend	158
57	Clock Count Register (CCNT).....	158
58	Performance Monitor Count Register (PMN0 - PMN3)	158
59	Performance Monitor Control Register	159
60	Interrupt Enable Register	159
61	Overflow Flag Status Register	160
62	Event Select Register	161
63	Performance Monitoring Events	162
64	Common Uses of the PMU.....	163
65	Multiply with Internal Accumulate Format	170
66	MIA{ <cond> } acc0, Rm, Rs.....	171
67	MIAPH{ <cond> } acc0, Rm, Rs.....	171
68	MIAXy{ <cond> } acc0, Rm, Rs	172
69	Internal Accumulator Access Format	173
70	MAR{ <cond> } acc0, RdLo, RdHi	174
71	MRA{ <cond> } RdLo, RdHi, acc0	174
72	First-Level Descriptors	175
73	Second-Level Descriptors for Coarse Page Table	175
74	Second-Level Descriptors for Fine Page Table	176
75	Exception Summary.....	177
76	Event Priority	177
77	Processors': Encoding of Fault Status for Prefetch Aborts	178
78	Intel XScale® Processor Encoding of Fault Status for Data Aborts.....	179
79	Branch Latency Penalty	182



80	Latency Example.....	184
81	Branch Instruction Timings (Those Predicted by the BTB)	184
82	Branch Instruction Timings (Those not Predicted by the BTB).....	184
83	Data Processing Instruction Timings	184
84	Multiply Instruction Timings.....	185
85	Multiply Implicit Accumulate Instruction Timings	187
86	Implicit Accumulator Access Instruction Timings.....	187
87	Saturated Data Processing Instruction Timings.....	187
88	Status Register Access Instruction Timings	187
89	Load and Store Instruction Timings	187
90	Load and Store Multiple Instruction Timings.....	188
91	Semaphore Instruction Timings.....	188
92	CP15 Register Access Instruction Timings.....	188
93	CP14 Register Access Instruction Timings.....	188
94	Exception-Generating Instruction Timings	189
95	Count Leading Zeros Instruction Timings.....	189
96	Pipelines and Pipe Stages.....	191
97	Processors: Network Processor Functions	222
98	Bus Arbitration Example: Three Requesting Masters	226
99	Memory Map	226
100	Register Legend.....	240
101	Ethernet MAC 0 on NPE B.....	240
102	Ethernet MAC 1 on NPE B.....	241
103	Ethernet MAC 2 on NPE B.....	243
104	Ethernet MAC 2 on NPE B.....	244
105	Ethernet MAC on NPE A.....	246
106	Ethernet MAC on NPE C	247
107	Endpoint Configuration: Universal Serial Bus Device Controller	280
108	USB States.....	281
109	Endpoint Field Addressing	283
110	IN, OUT, and SETUP Token Packet Format.....	284
111	SOF Token Packet Format	284
112	Data Packet Format.....	285
113	Handshake Packet Format	285
114	Bulk Transaction Formats.....	286
115	Isochronous Transaction Formats	286
116	Control Transaction Formats	286
117	Interrupt Transaction Formats	287
118	Host Device Request Summary	288
119	Register Legend.....	290
120	USB-Device Register Descriptions.....	290
121	Configuration Controls	363
122	Register Legend.....	364
123	Interface Register Sets	365
124	Host Capability Registers	366
125	Identification Register Fields.....	367
126	HWGENERAL – General Hardware Parameters: Fields.....	368
127	HWHOST – Host Hardware Parameters.....	368
128	HWDEVICE – Device Hardware Parameters	369
129	HWTXBUF – TX Buffer Hardware Parameters.....	369
130	HWRXBUF – RX Buffer Hardware Parameters	370
131	HCSPARAMS – Host Control Structural Parameters	371
132	HCCPARAMS – Host Control Capability Parameters	372
133	DCCPARAMS - Device Control Capability Parameters	373
134	USBCMD – USB Command Register	374



135 USBSTS – USB Status	376
136 USBINTR – USB Interrupt Enable	377
137 FRINDEX – USB Frame Index	379
138 PERIODICLISTBASE - Host Controller Frame List Base Address	380
139 ASYNCLISTADDR - Host Controller Next Asynchronous Address	380
140 BURSTSIZE - Host Controller Embedded TT Async. Buffer Status	381
141 PORTSCx - Port Status Control[1:8]	383
142 USBMODE - USB Device Mode	387
143 Typ Field Value Definitions	389
144 Next Schedule Element Pointer	393
145 iTD Transaction Status and Control	393
146 iTD Buffer Pointer Page 0 (Plus)	394
147 iTD Buffer Pointer Page 1 (Plus)	394
148 iTD Buffer Pointer Page 2 (Plus)	395
149 iTD Buffer Pointer Page 3-6	395
150 Next Link Pointer	396
151 Endpoint and Transaction Translator Characteristics	396
152 Micro-Frame Schedule Control	397
153 siTD Transfer Status and Control	397
154 Buffer Page Pointer List (Plus)	398
155 siTD Back Link Pointer	399
156 qTD Next Element Transfer Pointer (DWord 0)	400
157 qTD Alternate Next Element Transfer Pointer (DWord 1)	400
158 qTD Token (DWord 2)	401
159 qTD Buffer Pointer(s) (DWords 3-7)	403
160 Queue Head DWord 0	405
161 Endpoint Characteristics: Queue Head DWord 1	405
162 Endpoint Capabilities: Queue Head DWord 2	406
163 Current qTD Link Pointer	407
164 Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)	408
165 FSTN Normal Path Pointer Signals	409
166 FSTN Back Path Link Pointer Signals	409
167 Default Values of Operational Register Space	410
168 Default Port Routing Depending on EHCI HC CF Bit	412
169 Port Power Enable Control Rules	415
170 Behavior During Wake-Up Events	418
171 Example Worst-Case Transaction Timing Components	421
172 Operation of FRINDEX and SOFV (SOF Value Register)	424
173 Asynchronous Schedule State Machine Transition Actions	437
174 Typical Low- /Full-Speed Transaction Times	437
175 NakCnt Field Adjustment Rules	439
176 Actions for Park Mode, Based on Endpoint Response and Residual Transfer State	448
177 Example Periodic Reference Patterns for Interrupt Transfers with 2-ms Poll Rate	451
178 Ping Control State Transition Table	452
179 Ping State Encoding	452
180 Interrupt IN/OUT Do Complete Split State Execution Criteria	467
181 Initial Conditions for OUT siTD's TP and T-count Fields	475
182 Transaction Position (TP)/Transaction Count (T-Count) Transition Table	475
183 Summary siTD Split Transaction State	478
184 Example Case 2a - Software Scheduling siTDs for an IN Endpoint	480
185 Summary of Transaction Errors	483
186 Summary Behavior of EHCI Host Controller on Host System Errors	486
187 Standard EHCI vs. EHCI with Embedded Transaction Translator	487
188 Condition vs. Emulate TT Response	489
189 PCI Target Interface Supported Commands	498



190	PCI Initiator Interface Supported Commands	499
191	PCI Memory Map Allocation	508
192	PCI Byte Enables Using CRP Access Method	511
193	PCI Configuration Space.....	511
194	Command Type for PCI Controller Configuration and Status Register Accesses	512
195	PCI Target Interface Supported Commands	523
196	PCI Initiator Interface Supported Commands	525
197	PCI CLOCK and RESET Sourcing.....	528
198	PCI Byte Enables for Sub-word Single AHB Read/write Cycles	536
199	Register Legend.....	549
200	PCI Configuration Register Map.....	550
201	CSR Address Map.....	558
202	DDR1 SDRAM Memory Configuration Options.....	588
203	Supported DDR1 SDRAM Configurations	590
204	DDR1 SDRAM Address Register Summary	591
205	Address Decoding for DDR1 SDRAM Memory Banks	591
206	Programming Codes for the DDR1 SDRAM Bank Size	591
207	Programming Values for the DDR1 SDRAM 32-bit Size Register (S32SR[29:20])*	592
208	DDR1 SDRAM Address Translation for 128/512 Mbit (x16/x8), 1 Gbitx8, and 256 Mbitx8 Devices	593
209	DDR1 SDRAM Address Translation for 256 Mbitx16 Devices	593
210	DDR1 SDRAM Address Translation for 1 Gbitx16 Devices.....	594
211	DDR1 SDRAM Commands	598
212	Typical Refresh Frequency Register Values	614
213	Syndrome Decoding	620
214	MCU Error Response	627
215	Register Legend.....	630
216	Memory Controller Register Table.....	630
217	Example Expansion Bus Pin Mappings to Target Devices	651
218	Supported AHB Commands.....	652
219	Trimmed Version of IXP45X/IXP46X network processors Memory Map	653
220	Expansion Bus Address and Data Byte Steering.....	656
221	Multiplexed Output Pins for HPI Operation	664
222	HPI HCNTL Control Signal Decoding.....	665
223	Supported Inbound Expansion Bus Transfers	682
224	EX_ADDR Operation	685
225	Register Legend.....	701
226	Legacy Expansion Bus Register Summary	702
227	Non-Legacy Expansion Bus Register Summary	702
228	Bit Level Definition for each of the Timing and Control Registers.....	705
229	Configuration Register 0 Description	707
230	Setting The Intel XScale® Processor Operation Speed	709
231	Expansion Bus Configuration Register 1-Bit Definition	710
232	UTOPIA/Ethernet Configuration Options	715
233	NPE-B Ethernet Configuration Options.....	715
234	NPE-C Ethernet Configuration Options.....	715
235	EX_ADDR Value to Access EXP_INBOUND_ADDR Register.....	719
236	EX_ADDR Value to Access EXP_LOCK0 Register.....	720
237	EX_ADDR Value to Access EXP_LOCK1 Register.....	721
238	HSS Tx/Rx Clock Output	735
239	HSS Tx/Rx Clock Output Frequencies and PPM Error	735
240	HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization.....	735
241	HSS Frame Output Characterization.....	736
242	Jitter Definitions	736
243	Timeslot Configurations	741



244 Typical Baud-Rate Settings	753
245 UART Transmit Parity Operation	755
246 UART Receive Parity Operation	755
247 UART Word-Length Select Configuration	755
248 UART FIFO Trigger Level.....	761
249 Register Legend	761
250 UART Registers Overview	762
251 UART IDD Bit Mapping	767
252 Register Legend	779
253 Register Summary	779
254 Occurrence Events.....	789
255 Duration Events	790
256 Register Legend	793
257 PMU Register Table.....	793
258 AHB North PMU Mapping	798
259 AHB South PMU Mapping	798
260 Event Mux Programming	799
261 Intel XScale® Processor Interrupt Mapping	804
262 Register Legend	811
263 Interrupt Controller Memory Mapped Registers	811
264 Register Legend	821
265 Register Summary	821
266 System Time Clock Rates	834
267 Register Legend	836
268 Register Summary Table	837
269 Register Summary	838
270 Texas Instruments* Synchronous Serial Frame Format.....	859
271 Motorola* SPI Frame Format.....	861
272 National Microwire* Frame Format.....	862
273 SSP Serial Port Register Summary	863
274 Register Legend	864
275 Motorola* SPI Frame Formats for SPO and SPH Programming.....	867
276 I ² C Bus Definitions	876
277 Modes of Operation.....	878
278 START and STOP Bit Definitions.....	879
279 Master Transactions.....	887
280 Slave Transactions.....	890
281 General Call Address Second Byte Definitions	892
282 Register Legend	897
283 I ² C Register Addresses.....	897
284 PKE Peripheral Memory Map and Access Information	908
285 Register Legend	911
286 Register Summary	911
287 EAU Operand Size Restrictions and Assumptions.....	914
288 Register Legend	915
289 Register Summary	915
290 EAU RAM Memory Locations.....	919
291 Register Legend	921
292 Hashing Coprocessor: Register Summary.....	921
293 AHB Queue Manager Memory Map	928
294 Queue Status Flags.....	934
295 Data Validity Cases and Their Handling.....	936
296 Register Legend	937
297 Register Summary	937
298 NPE Coprocessor Error	953



299 NPE Coprocessor Response.....	955
300 NPE Reset State.....	956



Revision History

Date	Revision	Description
August 2006	004	Section 2.1.1, Section 2.1.3, Section 4.0: Updated the number of supported SMII ports from six to three. (CCR 1977120) Section 3.8.2.1: Updated endian operation information (CCR1973155) Table 99, “Memory Map”: Updated notes (CCR1975264) Section 6.1.3, Section 6.2.52, Table 105, and Table 106: Added clarifying information regarding the MDI Interface (CCR1970478) Section 11.2.1: Updated software read/write information (CCR1977819) Section 11.2.2.8: Updated initialization information Section 12.1: Updated overview paragraph (CCR2076121) Section 12.5.11: Clarified software disable feature (CCR1976679) Section 15.4: Updated GPIO example (CCR2039084) Section 21.5.1: Added note to clarify SCL operation (CCR2002036) Removed SS-SMII references since this feature is not supported (CCR1980266) Updated ‘Intel® XScale Core’ references to be ‘Intel XScale® Processor’. Only the terminology used in the document has changed. Incorporated specification changes, specification clarifications and document changes from the <i>Intel® IXP4XX Product Line of Network Processors Specification Update</i> (306428-004 and 306428-005)
August 2005	003	Section 2.2.11: Corrected number of PMU 32-bit event counters to 4. [SCR4324] Figure 33 and Figure 34: Corrected UTP_OP_ADDR and UTP_IP_ADDR values in UTOPIA polling illustrations. [SCR4323] Section 11.2.3.5: Clarified MCU behavior when ECC disabled. [SCR4303] Section 12.4.1.5: Added new description and figures for using I/O wait. Table 99 and Section 17.4: Corrected multiple text references for NPE A (interrupt 0), NPE B (interrupt 1), and NPE C (interrupt 2). [SCR4322] Section 16.3.4.3: Added descriptions for MCU PMU event programming. Section 16.6.1 through Section 16.6.5: Corrected physical address values for ESR0, ESR1, PSR, PMR, PECx, and PMSR. [SCR4299] Table 260: Clarified MCU event types (0 = page miss, 1 = page hit) Section 19.4: Added new section: “Theory of Operation (Auxiliary Snapshots)”. Section 25.3.1: Added new section: “Operand Restrictions”.
May 2005	002	Added support for Intel® IXP455 Network Processor, including Figure 3 on page 47 and Table 97 on page 222. Reordered chapters 6-28 to align with Chapter 2.0 topic order. See the <i>Intel® IXP4XX Product Line of Network Processors Specification Update</i> for chapter numbering from/to list. Chapter 12.0, “Expansion Bus Controller” changed text from IXP4XX processor to IXP425 processor. Chapter 13.0, “HSS Coprocessor” corrected signal names in certain timing diagrams (change bars in figure title only). Chapter 18.0, “Operating System Timer” added note about predictable operation of reload register. Chapter 20.0, “Synchronous Serial Port” enhanced descriptions of EFWR, STRF, RFL, TFL, and ROR bits.
March 2005	001	Initial release of document.





1.0 Introduction

1.1 About This Document

This document is the authoritative and definitive reference for the external architecture of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors based on Intel XScale® Technology.

1.2 Intended Audience

This document is intended to be used by:

- Software engineers needing to program at the register or assembly level
- System architects needing to ensure that a system will operate and perform to specification
- Board designers to use for critical functional interfaces and bootstrapping option for the part
- Anyone else seeking detailed knowledge on the functional interworkings of the part

1.3 How to Read This Document

Each chapter in this document focuses on a specific architectural feature of the IXP46X network processors.

1.4 Other Relevant Documents

Document Title	Document #
Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet	306261
Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Developer's Manual	306262
Intel® IXP4XX Product Line of Network Processors Specification Update	306428
Intel® IXP400 Software Programmer's Guide	252539
Intel® XScale™ Microarchitecture Programmer's Reference Manual	273436
PCI Local Bus Specification, Revision 2.2	
Universal Serial Bus Specification, Revision 1.1	
Universal Serial Bus Specification, Revision 2.0	
UTOPIA Level 2 Specification, Revision 1.0	
IEEE 802.3 Specification	
IEEE 1149.1 Specification	
I2C Specification	
SSP Specification	
SMII Specification	



1.5 Terminology and Conventions

This section explains the naming conventions and the terminology used in the document. Common acronyms are defined in [Table 1](#).

Table 1. List of Acronyms (Sheet 1 of 4)

Acronym	Description
AALx	ATM Adaptation Layer x (0, 2, 5)
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
AQM	AHB Queue Manager
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BERT	Bit-Error Rate Test
BFA	Basic Frame Alignment
BIU	Bus Interface Unit
Bps	Bytes per second
bps	Bits per second
BSD	Berkeley Software Design
BSP	Board-Support Package
BTB	Branch Target Buffer
CBR	Constant Bit Rate
CPE	Customer Premise Equipment
CPID	Connection Point Identifier
CPU	Central Processing Unit
CRC	Cyclical Redundancy Check
DDR1 SDRAM	Double-Data Rate SDRAM, Version 1 (266 MHz)
DES	Data-Encryption Standard
DMA	Direct Memory Access
DRAM	Dynamic, Random-Access Memory
DSA	Digital Signature Algorithm
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
DSP	Digital Signal Processor
E1	Euro 1 trunk line (2.048 Mbps)
EMI	Electro-Magnetic Interference
FCS	Frame-Check Sequence
FIFO	First-In, First-Out
FRAM	Fixed Random Access Memory



Table 1. List of Acronyms (Sheet 2 of 4)

Acronym	Description
GBps	Gigabytes per second
Gbps	Gigabits per second
GCI	General Circuit Interface
GPIO	General-Purpose Input/Output
HDLC	High-Level Data Link Control
HEC	Header-Error Check
HMAC	Hashed Message Authentication Code
HPI	Host-Port Interface
HSS	High-Speed Serial (port)
HW	Hardware
I/O	In/Out
IAD	Integrated Access Device
IBPMU	Internal Bus Performance Monitoring Unit
IC	Integrated Circuit
ICE	In-Circuit Emulator
IEEE	Institute of Electrical and Electronic Engineers
IKE	Internet Key Exchange
IMA	Inverse Multiplexing over ATM
IOM	ISDN Orientated Modular
IP	Internet Protocol
IPsec	Internet Protocol Security
IRQ	Interrupt Request
ISA	Instruction Set Architecture
ISDN	Integrated Services Digital Network
ISR	Interrupt Service Routine
ITU	International Telecommunication Union
IXA	Internet Exchange Architecture
IXP	Internet Exchange Processor
JTAG	Joint Test Action Group
LAN	Local Area Network
LED	Light-Emitting Diode
LFSR	Linear Feedback Shift Register
LLC	Logical Link Control
LLC/SNAP	Logical Link Control / Subnet Access Protocol
LSb	Least-Significant Bit
LSB	Least-Significant Byte
LUT	Look-Up Table
MAC	Media Access Controller
Mbps	Megabits per second
MBps	Megabytes per second

**Table 1. List of Acronyms (Sheet 3 of 4)**

Acronym	Description
MD5	Message Digest 5
MDIO	Management Data Input/Output
MFAS	Multiframe Alignment Signals
MFS	Maximum Frame Size
MIB	Management Information Base
MII	Media-Independent Interface
MIPS	Million Instructions Per Second
MLPPP	Multi-Link, Point-to-Point Protocol
MMU	Memory Management Unit
MPHY	Multi PHY
MSb	Most-Significant Bit
MSB	Most-Significant Byte
MVIP	Multi-Vendor Integration Protocol
NEG	Negative
NMI	Nonmaskable Interrupt
NPE	Network Processor Engine
NRT-VBR	Non-Real-Time Variable Bit Rate
NRZI	Non-Return To Zero Inverted
OC 3	Optical Carrier 3
OS	Operating System
PBGA	Plastic-Ball Grid Array
PCB	Printed Circuit Board
PCI	Peripheral Component Interface
PCM	Pulse Code Modulation
PEC	Programmable-Event Counters
PDU	Protocol Data Unit
PHY	Physical Layer (Layer 1) Interface
PMU	Performance Monitoring Unit
PRD	Product Requirements Document
PSM PSM2	Outdated terminology for the NPE (Network Processor Engine) microprocessor core. The preferred term is NPE core.
QMgr	Queue Manager
RAM	Random-Access Memory
rt-VBR	Real-Time Variable Bit Rate
Rx	Receive
SAR	Segmentation and Reassembly
SDRAM	Synchronous Dynamic Random Access Memory (See "DDR1 SDRAM")
SFD	Start-of-Frame Delimiter
SHA1	Secure Hash Algorithm 1
SMII	Serial Media Independent Interface

**Table 1. List of Acronyms (Sheet 4 of 4)**

Acronym	Description
SNMP	Simple Network Management Protocol
SOF	Start-of-Frame
SPHY	Single PHY
SRAM	Static Random Access Memory
SSRAM	Synchronous Static Random Access Memory
T1	Type 1 (trunk line)
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDM	Time Division Multiplexing
TLB	Time slot Loopback Translation Look-Aside Buffer
TMS	Test-Mode Select
Tx	Transmit
UART	Universal Asynchronous Receiver-Transmitter
UBR	Unspecified Bit Rate
USB	Universal Serial Bus
UTOPIA	Universal Test and Operation PHY Interface for ATM
VBR	Variable Bit Rate
VC	Virtual Channel
VCC	Virtual Circuit Connection
VCI	Virtual Circuit Identifier
VDSL	Very-High-Speed, Digital Subscriber Line
VLAN	Virtual Local Area Network
VoIP	Voice-Over-Internet Protocol
VPN	Virtual Private Network
WAN	Wide Area Network
xDSL	Any Digital Subscriber Line
ZBT	Zero-bus Turn Around

1.5.1 Number Representation

All numbers in this document can be assumed to be base 10 unless designated otherwise. In text and pseudo code descriptions, hexadecimal numbers have a prefix of 0x and binary numbers have a prefix of 0b. For example, 107 would be represented as 0x6B in hexadecimal and 0b1101011 in binary.

1.5.2 Signal-Naming Convention

In this document, the following naming conventions are used:

- A word refers to 32 bits.
- Dword for PCI transactions, PCI specification refers to Dword as 32-bit transactions
- Signal names or register bit names ending with ‘_N’ or ‘#’ are active low, for example RTS_N.
Signals or register bits without this are active high.



1.5.3 Register Legend

In this document, the following register access definitions are used:

Table 2. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set



2.0 Functional Overview

The Intel® IXP45X and Intel® IXP46X Product Line of Network Processors are compliant with the Intel® StrongARM® Version 5TE instruction-set architecture (ISA). The IXP45X/IXP46X network processors are designed with Intel 0.18- μ semiconductor process technology. This process technology — along with the compactness of the Intel® StrongARM® RISC ISA, the ability to simultaneously process data with up to three integrated network processing engines (NPEs), and numerous dedicated-function peripheral interfaces — enables the IXP45X/IXP46X network processors to operate over a wide range of low-cost networking applications with industry-leading performance.

As indicated in [Figure 1](#), [Figure 2](#), and [Figure 3](#), the IXP45X/IXP46X network processors combine many features with the Intel XScale® Processor to create a highly integrated processor applicable to LAN/WAN-based networking applications in addition to other embedded networking applications.

This section briefly describes the main features of the product. For details on feature support listed by processor model, see the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*.

Figure 1. Intel® IXP465 Network Processor Block Diagram

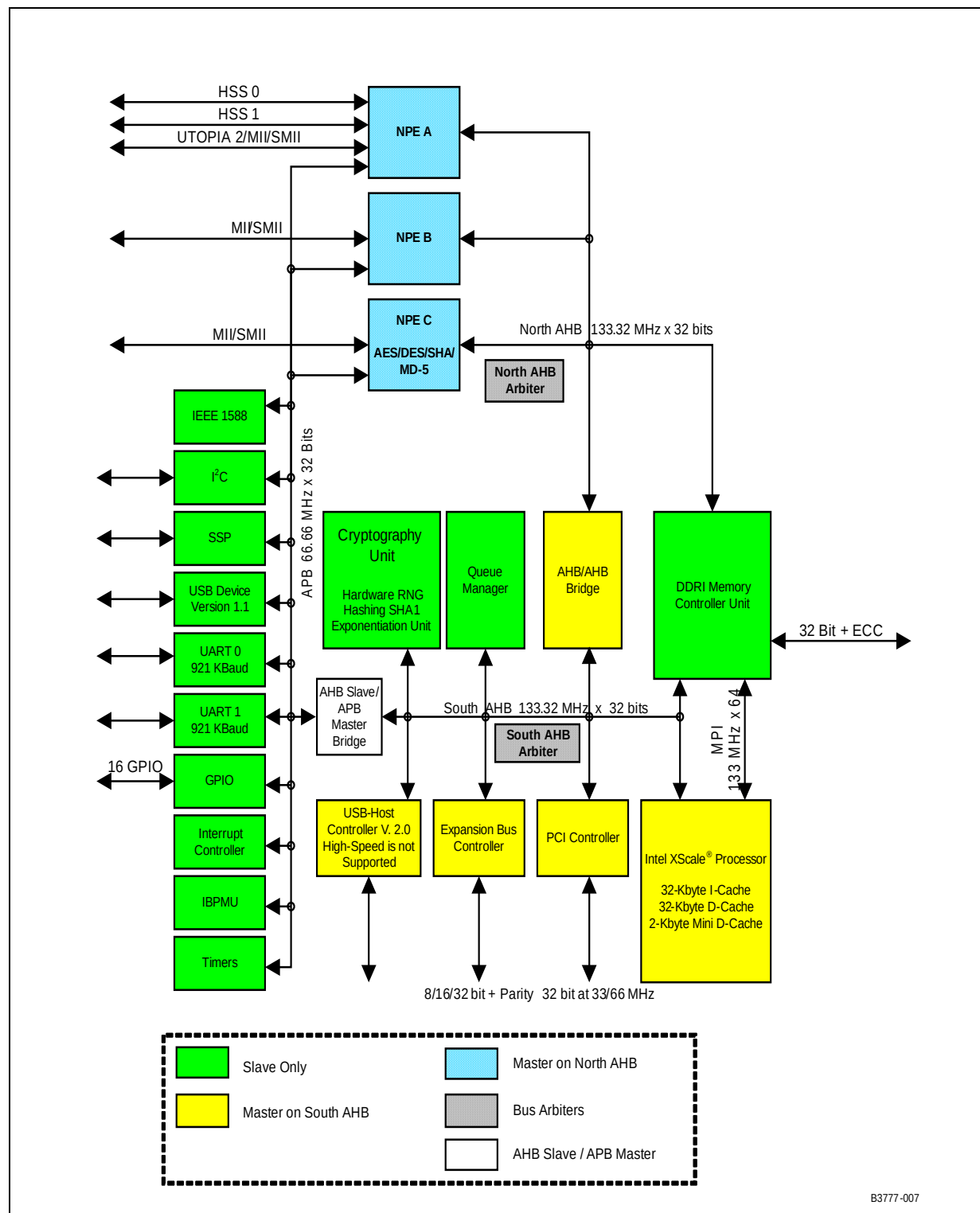
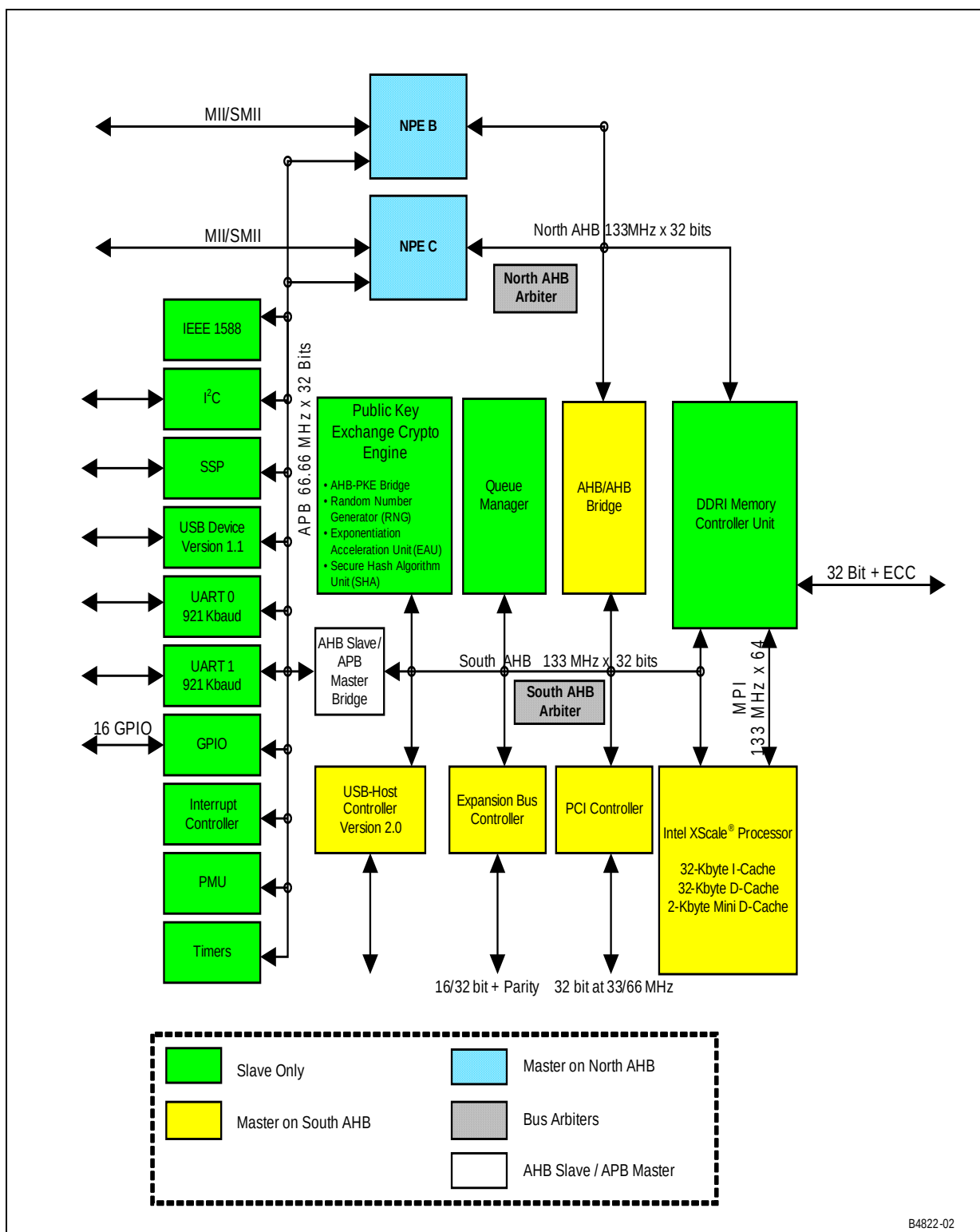




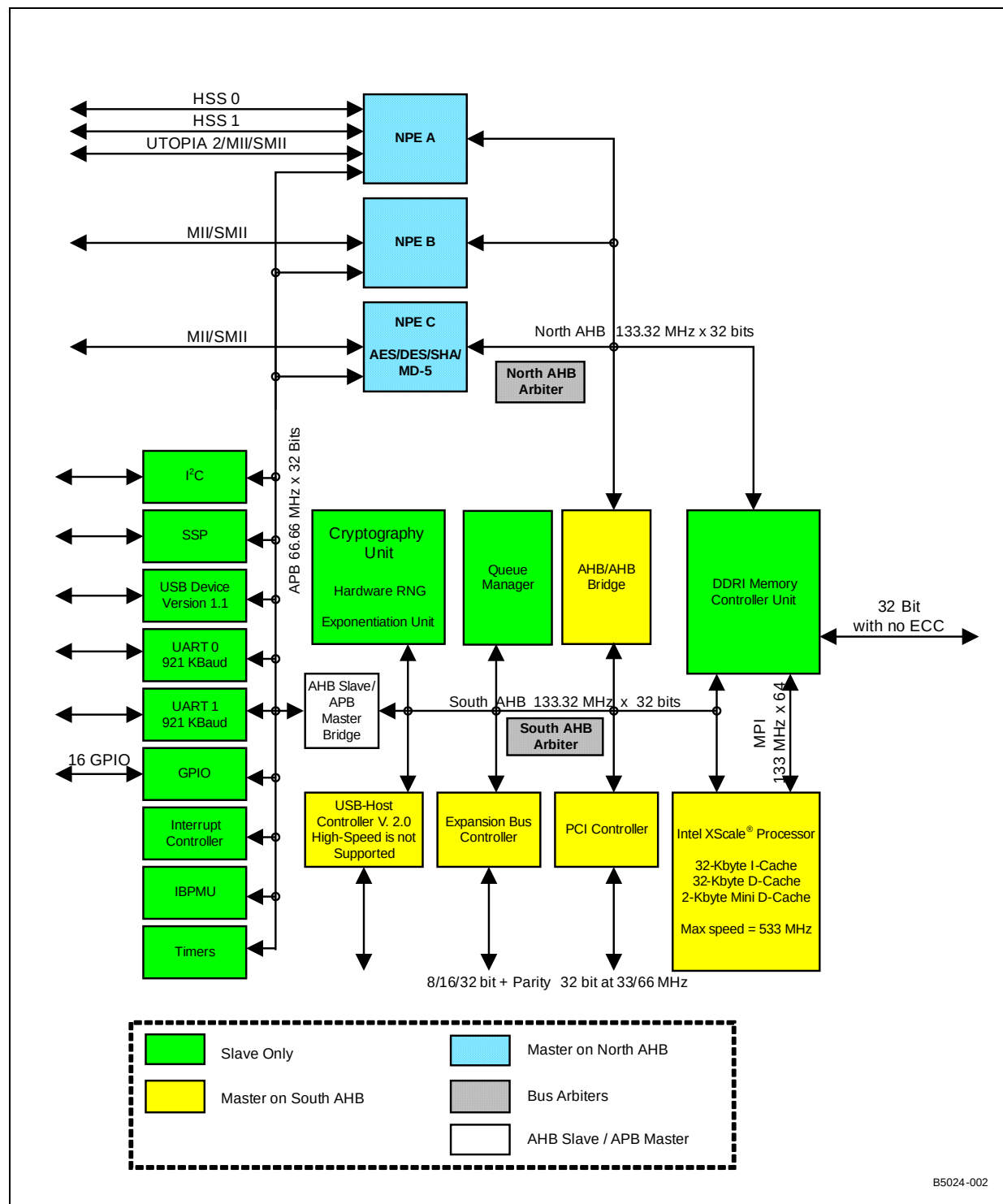
Figure 2. Intel® IXP460 Network Processor Block Diagram



B4822-02



Figure 3. Intel® IXP455 Network Processor Block Diagram



B5024-002



2.1 Key Functional Units

The following sections briefly describe the functional units and their interaction in the system.

Unless otherwise specified, the functional descriptions apply to all of the IXP45X/IXP46X network processors.

For details on feature support listed by processor model, see the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*.

2.1.1 Network Processor Engines (NPEs)

The network processor engines (NPEs) are dedicated-function processors containing hardware coprocessors integrated into the IXP45X/IXP46X network processors. The NPEs are used to off-load processing function required by the Intel XScale processor.

These NPEs are high performance, hardware multi-threaded processors with additional local hardware assist functionality used to off-load highly processing intensive functions such as MII (MAC), CRC checking/generation, AAL segmentation and re-assembly, AES, AES-CCM, DES, SHA-1/256/384/512, MD5, etc.

All instruction code is stored locally for the NPEs, each of which has a dedicated instruction memory bus and dedicated data memory bus.

These NPEs support processing of the dedicated peripherals that can include:

- One UTOPIA Level 2 (Universal Test and Operation PHY Interface for ATM) interface
- Two High-Speed Serial (HSS) interfaces
- Up to three Media-Independent Interface (MII) or Serial Media Independent Interfaces (SMII)

There are several possible combination of interfaces for the NPEs contained on the IXP45X/IXP46X network processors. These interface combinations are configured by setting expansion bus address straps during reset. Detailed information on settings can be found in the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*. Refer to the *Intel® IXP4XX Product Line of Network Processors Specification Update* as well to ensure the most up-to-date information regarding corrections to documentation.

The NPE core is a hardware multi-threaded processor engine separate from, but operating in parallel with the Intel XScale processor. The NPE is used to off-load and accelerate network data packet processing that would otherwise take processing cycle time on the Intel XScale processor. Each NPE core is a 133.32-MHz processor core that has self-contained instruction memory and self-contained data memory that operate in parallel. Each NPE core has 4 K words of instruction memory and 4 K words of data memory.

In addition to having separate instruction/data memory, the NPE core supports hardware multi-threading with support for multiple contexts. The support of hardware multi-threading creates an efficient processor engine with minimal processor stalls due to the ability of the processor to switch contexts in a single clock cycle, based on a prioritized/preemptive basis. The prioritized/preemptive nature of the context switching allows time-critical applications to be implemented in a low-latency fashion — which is required when processing multi-media applications.

The NPE core also connects to several hardware-based coprocessors that are used to implement functions that are difficult for a processor to implement. These functions include:



- HSS Serialization/ De-serialization
- DES/3DES/AES
- MD-5
- Learning/filtering content addressable memory
- UTOPIA Level 2 Framing
- CRC checking/generation
- SHA-1/256/384/512
- HDLC bit stuffing/de-stuffing
- Media Access Controller functionality
- DMA capability of data movement to and from DDRI memory

These coprocessors are implemented in hardware, enabling the coprocessors and the NPE processor core to operate in parallel.

With the addition of the new switching coprocessor (SWCP) and the Ethernet coprocessors, functions like a four-port, Layer-2 switch can be easily implemented using all Intel-based silicon. Also, by using NPEs to implement switching functions, value added features like VLAN or IP switching can be easily upgraded using existing silicon.

The combined forces of the hardware multi-threading, independent instruction memory, independent data memory, and parallel processing — contained on the NPE — allows the Intel XScale processor to be utilized for application purposes. The multi-processing capability of the peripheral interface functions allows unparalleled performance to be achieved by the application running on the Intel XScale processor.

Note: All the described NPE functions require Intel-supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.2 Internal Bus

The internal bus architecture of the IXP45X/IXP46X network processors are designed to allow parallel processing to occur and to isolate bus utilization, based on particular traffic patterns. The bus is segmented into four major buses:

- North - Advanced High Performance Bus (AHB)
- South - AHB
- Memory Port Interface
- Advanced Peripheral Bus (APB)

2.1.2.1 North AHB

The North AHB is a 133.32-MHz, 32-bit bus that can be mastered by the NPE A, NPE B, or NPE C. The targets of the North AHB can be the DDRI memory controller or the AHB/AHB bridge. The AHB/AHB bridge allows the NPEs to access the peripherals and internal targets on the South AHB.

Data transfers by the NPEs on the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB bridge may be “posted” when writing or “split” when reading.

When a transaction is “posted,” a master on the North AHB requests a write to a peripheral on the South AHB. If the AHB/AHB Bridge has a free FIFO location, the write request will be transferred from the master on the North AHB to the AHB/AHB bridge.



The AHB/AHB bridge will complete the write on the South AHB, when it can obtain access to the peripheral on the South AHB. The North AHB is released to complete another transaction.

When a transaction is “split,” a master on the North AHB requests a read of a peripheral on the South AHB. If the AHB/AHB bridge has a free FIFO location, the read request will be transferred from the master on the North AHB to the AHB/AHB bridge. The AHB/AHB bridge will complete the read on the South AHB, when it can obtain access to the peripheral on the South AHB.

Once the AHB/AHB bridge has obtained the read information from the peripheral on the South AHB, the AHB/AHB bridge notifies the arbiter, on the North AHB, that the AHB/AHB bridge has the data for the master that requested the “split” transfer. The master on the North AHB which requested the split transfer will arbitrate for the North AHB and transfer the read data from the AHB/AHB bridge. The North AHB is released to complete another transaction while the North AHB master that requested the “split” transfer waits for the data to arrive.

These “posting” and “splitting” transfers allow control of the North AHB to be given to another master on the North AHB — enabling the North AHB to achieve maximum efficiency. Transfers to the AHB/AHB bridge are assumed to be small and infrequent, relative to the traffic passed between the NPEs on the North AHB and the DDRI SDRAM.

Arbitration on the North AHB is round-robin. Each transaction can be no longer than an eight-word bursts. This implementation promotes fairness within the system.

2.1.2.2 South AHB

The South AHB is a 133.32-MHz, 32-bit bus that can be mastered by the Intel XScale® Processor, PCI controller, Expansion Bus Interface, USB Host Controller, and the AHB/AHB bridge. The targets of the South AHB Bus can be the DDRI SDRAM, PCI Controller, Queue Manager, Expansion Bus, Cryptography Unit, or the AHB/APB bridge.

Accessing across the APB/AHB bridge allows interfacing to peripherals attached to the APB. The Expansion bus is the only target on the South AHB Bus that can be configured to support split transfers.

Arbitration on the South AHB is round-robin. Each transaction can be no longer than an eight-word bursts. This implementation promotes fairness within the system.

2.1.2.3 Memory Port Interface

The Memory Port Interface provides a dedicated interface between Intel XScale® Processor to the DDRI SDRAM. The Memory Port Interface operates at 133 MHz. This implementation will allow a theoretical maximum bus bandwidth of over 1 GBps performance.

The Memory Port Interface stores memory transactions from the Intel XScale processor which have not been processed by the Memory Controller. The Memory Port Interface supports eight Intel XScale processor read transactions up to 32 bytes each. That total equals the maximum number of outstanding transaction the Processor Bus Controller can support (That includes core DCU [4 - load requests to unique cache lines], IFU [2 - prefetch], IMM [1 - tablewalk], DMM [1 - tablewalk].)

The Memory Port Interface also supports eight Intel XScale processor-posted write transactions up to 16 bytes each.

Arbitration on the Memory Port Interface is not required due to no contention with other masters. Arbitration will exist in the DDRI memory controller between all of the main internal interfaces.



2.1.2.4 APB Bus

The APB Bus is a 66.66-MHz, 32-bit bus that can be mastered by the AHB/APB bridge only. The targets of the APB bus can be:

- USB 1.1 device controller
- The internal bus performance monitoring unit
- GPIO
- IEEE 1588 Hardware Assist
- I²C
- UARTs
- All NPEs
- Interrupt controller
- Timers
- Synchronous Serial Protocol Interface

The APB interface is also used for NPE code download and configuration.

No arbitration is required due to a single master implementation.

2.1.3 MII/SMII Interfaces

The IXP45X/IXP46X network processors can be configured to support up to three MII or SMII industry-standard, media-independent interface (MII) interfaces. These interfaces are integrated into the IXP45X/IXP46X network processors with separate media-access controllers and in many cases independent network processing engines.

The independent NPEs and MACs allow parallel processing of data traffic on the MII interfaces and off-loading of processing required by the Intel XScale processor. The IXP45X/IXP46X network processors are compliant with the IEEE 802.3 specification.

In addition to the MII interfaces, the IXP45X/IXP46X network processors include a single management data interface that is used to configure and control PHY devices that are connected to the MII interfaces. The IXP45X/IXP46X network processors provide support for the serial media independent interface (SMII).

Note: All the described NPE functions require Intel-supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.4 UTOPIA Level 2

The integrated UTOPIA Level 2 interface works with a network-processing engine for several of the IXP45X/IXP46X network processors. The pins of the UTOPIA Level 2 interface are multiplexed with one of the MII/SMII interfaces.

The UTOPIA Level 2 interface supports a single- or a multiple-physical-interface configuration with cell-level or octet-level handshaking. The network processing engine handles segmentation and reassembly of ATM cells, CRC checking/generation, and transfer of data to/from memory. This allows parallel processing of data traffic on the UTOPIA Level 2 interface, off-loading processor overhead required by the Intel XScale processor.

The IXP45X/IXP46X network processors are compliant with the ATM Forum's *UTOPIA Level 2 Specification*, Revision 1.0. The UTOPIA Level 2 interface of the IXP45X/IXP46X network processors is an 8-bit interface.



Note: All the described NPE functions require Intel-supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.5 Universal Serial Bus (USB) Interfaces

There are two USB interfaces supported by the IXP45X/IXP46X network processors:

- USB 1.1 Device Interface
- USB 2.0 Host Interface

2.1.5.1 USB 1.1 Device Interface

The integrated USB 1.1 device interface supports full-speed operation and 16 endpoints and includes an integrated transceiver.

There are:

- Six isochronous endpoints (three input and three output)
- One control endpoints
- Three interrupt endpoints
- Six bulk endpoints (three input and three output)

2.1.5.2 USB 2.0 Host Interface

This unit supplies USB Host functionality. The function being performed is defined by the USB 2.0 specification, maintained by usb.org, and the interface is (largely) EHCI-compliant, as defined by Intel.

Not all features defined by the 2.0 specification are supported for this implementation. The following is a partial list of supported features:

- Host function
- Low speed interface
- Full speed interface
- EHCI register interface

The following is a partial list of features *not* supported:

- Device function
- OTG function
- High speed interface

2.1.6 PCI Controller

The PCI controller of the IXP45X/IXP46X network processors is compatible with the *PCI Local Bus Specification*, Rev. 2.2. The PCI interface is 66Mhz 32-bit compatible bus and capable of operating as either a host or an option.

When the PCI Controller is configured as a host, an internal PCI arbiter may be utilized to allow up to four devices to be connected to the IXP45X/IXP46X network processors without the need for an external arbiter.

Features of the PCI Controller include:

- Conforms to PCI Local Bus Specification Revision 2.2



- 32-bit, 0-66MHz PCI bus operation
- Provides initiator (master) and target (slave) PCI interfaces
- Provides AHB-to-PCI and PCI-to-AHB DMA channels
- Includes PCI bus arbiter supporting up to 4 external PCI masters using round-robin arbitration.
- Access to PCI Configuration registers from PCI and AHB busses
- Provides interrupt to processor to indicate transaction errors on the PCI or AHB bus
- Provides interrupt to processor for DMA complete and DMA error
- Provides doorbell interrupt generation capability to PCI and AHB agents
- Byte, half word, word single reads/writes, burst word reads/writes supported on AHB and busses
- Generates memory, I/O, and configuration cycles as PCI master
- Provides AHB masters with full access to the 4Gbyte PCI address space
- Provides PCI-to-AHB address translation to map PCI accesses to AHB address space

2.1.7 DDRI SDRAM Controller

The IXP45X/IXP46X network processors integrate a high-performance, multi-ported Memory Controller Unit (MCU) to provide a direct interface between the IXP45X/IXP46X network processors and their local memory subsystem. The MCU supports:

- DDRI 266 SDRAM
- 128/256/512-Mbit, 1-Gbit DDRI SDRAM technology support
- Only unbuffered DRAM support (No registered DRAM support)
- Dedicated port for Intel XScale® Processor to DDRI SDRAM
- 32 Mbyte 32-bit DDRI SDRAM for low-cost solutions
- Up to 1 Gbyte of 32-bit DDRI SDRAM for large memory requirements
- Single-bit error correction, multi-bit detection support (ECC)
- 32, 40-bit wide Memory Interfaces (non-ECC and ECC support)

The DDRI SDRAM interface provides a direct connection to a high bandwidth and reliable memory subsystem. The DDRI SDRAM interface consists of a 32-bit-wide data path to support up to 1.0 GBps throughput.

An 8-bit Error Correction Code (ECC) across each 32-bit word improves system reliability. It is important to note that ECC is also referred to as CB in many DIMM specs. The pins used for ECC on the IXP45X/IXP46X network processors are called `DDRI_CB[7:0]`. The controller supports the 8 bits due to the fact that internally it is a 32 or 64 bit controller. However, this implementation of the controller only supports 32 bits. The ECC circuitry was designed to operate always on a 64 bit word and when operating in 32 bit mode, the upper 32 bits are driven to zeros internally.

Note: The Intel® IXP455 Network Processor does not support ECC functionality. For details, see the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*.

The MCU supports two banks of DDRI SDRAM. The MCU has support for unbuffered DDRI 266 only.



Table 3 on page 54 illustrates the supported DDRI SDRAM configurations for the IXP45X/IXP46X network processors. The 128/256/512-Mbit, 1-Gbit DDRI SDRAM devices comprise four internal leaves. The MCU controls the leaf selects within 128/256/512-Mbit, 1-Gbit DDRI SDRAM by toggling DDRI_BA[0] and DDRI_BA[1].

The two DDRI SDRAM chip enables DDRI_CS_N[1:0] support a DDRI SDRAM memory subsystem consisting of two banks. The base address for the two contiguous banks are programmed in the DDRI SDRAM Base Register (SDBR) and must be aligned to a 32-Mbyte boundary. The size of each DDRI SDRAM bank is programmed with the DDRI SDRAM boundary registers (SBR0 and SBR1).

Table 3. Supported DDRI Memory Configurations

DDR1 SDRAM Technology	DDR1 SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size ¹
			Row	Col	DDR1_BA[1]	DDR1_BA[0]	
128 Mbit	16M x 8	1	12	10	I_AD[26]	I_AD[25]	64 Mbyte
		2					128 Mbyte
	8M x 16	1	12	9	I_AD[25]	I_AD[24]	32 Mbyte
		2					64 Mbyte
256 Mbit	32M x 8	1	13	10	I_AD[27]	I_AD[26]	128 Mbyte
		2					256 Mbyte
	16M x 16	1	13	9	I_AD[26]	I_AD[25]	64 Mbyte
		2					128 Mbyte
512 Mbit	64M x 8	1	13	11	I_AD[28]	I_AD[27]	256 Mbyte
		2					512 Mbyte
	32M x 16	1	13	10	I_AD[27]	I_AD[26]	128 Mbyte
		2					256 Mbyte
1 Gbit	128M x 8	1	14	11	I_AD[29]	I_AD[28]	512 Mbyte
		2					1 Gbyte
	64M x 16	1	14	10	I_AD[28]	I_AD[27]	256 Mbyte
		2					512 Mbyte
Notes:							
1. Table indicates 32-bit-wide memory subsystem sizes							

The memory controller is a 32-bit only interface. If a x16 memory chip is used, a minimum of two memory chips would be required to facilitate the 32-bit interface required by the IXP45X/IXP46X network processors. If ECC is required, additional memories would need to be added. For more information on DDRI SDRAM support and configuration see the Memory Controller section contained later in this document.

The memory controller internally interfaces to the North AHB, South AHB, and Memory Port Interface with independent interfaces. This architecture allows DDRI SDRAM transfers to be interleaved and pipelined to achieve maximum possible efficiency.

2.1.8 Expansion Interface

The expansion interface allows easy and — in most cases — glue-less connection to peripheral devices. It also provides input information for device configuration after reset.



Some of the peripheral device types are SRAM, flash, ATM control interfaces, and DSPs used for voice applications. (Some voice configurations can be supported by the HSS interfaces and the Intel XScale processor, implementing voice-compression algorithms.)

The expansion interface functions in two modes of operation:

- Legacy (16-bit, data mode)
- Enhanced (32-bit, data mode)

In the legacy mode of operation, the expansion interface is a 16-bit interface that allows an address range of 512 bytes to 16 Mbytes, using 24 address lines for each of the eight independent chip selects.

Accesses to the expansion bus interface consists of five phases. Each of the five phases can be lengthened or shortened by setting various configuration registers on a per-chip-select basis. This feature allows the IXP45X/IXP46X network processors to connect to a wide variety of peripheral devices with varying speeds.

The expansion interface supports Intel or Motorola* microprocessor-style bus cycles. The bus cycles can be configured to be multiplexed address/data cycles or separate address/data cycles for each of the eight chip-selects.

Additionally, Chip Selects 4 through 7 can be configured to support Texas Instruments* HPI-8 or HPI-16 style accesses for DSPs.

The expansion interface is an asynchronous interface to externally connected chips. However, a clock must be supplied to the IXP45X/IXP46X network processors' expansion interface for the interface to operate. This clock can be driven from GPIO 15 or an external source. The maximum clock rate that the expansion interface can accept is 80 MHz.

By providing this legacy mode of operation, code developed for previous generations of this platform becomes easily portable.

In the enhanced mode of operation, the expansion interface is a 32-bit interface that allows an address range of 512 bytes to 32 Mbytes per chip select on the IXP45X/IXP46X network processors, using 25 address lines for each of the eight independent chip selects.

Additionally, in enhanced mode, the interface supports shared access to the internal bus with external masters. This shared access is achieved with four request/grant pins and an integrated arbiter. Not only can external devices access each other, but they can also access the IXP45X/IXP46X network processors' internal registers (including the DDRI SDRAM interface).

The advantage to this feature is that shared memory access can be achieved by using the DDRI SDRAM interface attached to the IXP45X/IXP46X network processors. This lowers the system's overall bill of materials.

Enhanced mode also supports synchronous transfers at speeds of up to 80 MHz with a 40-pF load. In addition to fully synchronous support, the enhanced mode also supports burst transfers of up to eight-word lengths. The synchronous bus support is compatible to Zero Bus Turnaround (ZBT) SRAM cycles for inbound/outbound transactions for both read/write transactions.

Additionally, the outbound read transactions can support the Intel StrataFlash® Embedded Memory P30 synchronous-burst support.

Byte-wide parity is an optional configuration of this interface in enhanced mode of operation.



At the de-assertion of reset, the address bus is used to capture configuration information from the levels that are applied to the pins at this time. External pull-up/pull-down resistors are used to tie the signals to particular logic levels. (For additional details, refer to the Expansion Bus Controller section of this document).

2.1.9 High-Speed, Serial Interfaces

The high-speed, serial interfaces (HSS) are six-signal interfaces that support serial transfer speeds from 512 KHz to 8.192 MHz, for some models of the IXP45X/IXP46X network processors. (For processor-specific speeds, refer to the HSS chapter.)

Each interface allows direct connection of up to four T1/E1 framers and CODEC/SLICs to the IXP45X/IXP46X network processors. The high-speed, serial interfaces are capable of supporting various protocols, based on the implementation of the code developed for the network processor engine.

For a list of supported protocols, see the *Intel® IXP400 Software Programmer's Guide*.

2.1.10 UARTs

The UART interfaces are a 16550-compliant UART with the exception of transmit and receive buffers. Transmit and receive buffers are 64 bytes-deep versus the 16 bytes required by the 16550 UART specification.

The interfaces can be configured to support speeds from 1,200 Baud to 921 Kbaud. The interfaces support configurations of:

- Five, six, seven, or eight data-bit transfers
- One or two stop bits
- Even, odd, or no parity

The request-to-send (RTS_N) and clear-to-send (CTS_N) modem control signals also are available with the interface for hardware flow control.

2.1.11 GPIO

There are 16 GPIO pins supported by the IXP45X/IXP46X network processors. GPIO pins 0 through 13 can be configured to be general-purpose input or general-purpose output. Additionally, GPIO pins 0 through 12 can be configured to be an interrupt input.

GPIO Pin 14 can be configured similar to GPIO Pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33 MHz, with various duty cycles. GPIO Pin 14 is configured as an input, upon reset.

GPIO Pin 15 can be configured the same as GPIO Pin 13 or as a clock output. The output-clock configuration can be set at various speeds, up to 33 MHz, with various duty cycles. GPIO Pin 15 is configured as an output, upon reset. GPIO Pin 15 can be used to clock the expansion interface, after reset.

Several other GPIO pins can serve as an alternate function. These functions are outlined in [Table 4](#) below.

**Table 4. GPIO Alternate Function Table**

GPIO Pin Number	Alternate Function
0	External USB 1.1 Device Clock input
1	External USB 2.0 Host Clock input
2	(Reserved)
3	(Reserved)
4	(Reserved)
5	(Reserved)
6	(Reserved)
7	Auxiliary IEEE1588 Master Snapshot input
8	Auxiliary IEEE1588 Slave Snapshot input
9:13	(Reserved)
14	Output Clock 14
15	Output Clock 15

2.1.12 Internal Bus Performance Monitoring Unit (IBPMU)

The IXP45X/IXP46X network processors consist of a performance monitoring unit that may be used to capture predefined events within the system outside of the Intel XScale processor. These features aid in measuring and monitoring various system parameters that contribute to the overall performance of the processor.

The Performance Monitoring (PMON) facility provided comprises:

- Eight Programmable Event Counters (PECx)
- Previous Master/Slave Register
- Event Selection Multiplexor

The programmable event counters are 27 bits wide. Each counter may be programmed to observe one event from a defined set of events. An event consists of a set of parameters which define a start condition and a stop condition.

The monitored events are selected by programming the Event Select Registers (ESR).

2.1.13 Interrupt Controller

The IXP45X/IXP46X network processors consist of up to 64 interrupt sources to allow an extension of the Intel XScale processor's FIQ and IRQ interrupt sources. These sources can originate from some external GPIO pins, internal peripheral interfaces, or internal logic.

The interrupt controller can configure each interrupt source as an FIQ, IRQ, or disabled. The interrupt sources tied to Interrupt 0 to 7 can be prioritized. The remaining interrupts are prioritized in ascending order. For example, Interrupt 8 has a higher priority than 9, 9 has a higher priority than 10, and 30 has a higher priority than 31.

An additional level of priority can be set for interrupts 32 through 63. This priority setting gives any interrupt between 32 through 63 priority over interrupts 0 through 31.



2.1.14 Timers

The IXP45X/IXP46X network processors consist of four internal timers operating at 66.66 MHz to allow task scheduling and prevent software lock-ups. The device has four 32-bit counters:

- Watch-Dog Timer
- Timestamp Timer
- Two general-purpose timers, read/writable reload

The Timestamp Timer and the two general-purpose timers have the optional ability to use a pre-scaled clock. A programmable pre-scaler can be used to divide the input clock by a 16-bit value. The input clock can be either the APB clock (66.66 MHz) or a 20-ns version of the APB clock (50 MHz). By default all timers use the APB clock.

The 16-bit pre-scale value ranges from divide by 2 to 65,536 and results in a new clock enable available for the timers that ranges from 33.33 MHz down to 1,017.26 Hz.

The Timestamp Timer also contains a 32-bit compare register that allows an interrupt to be created at times other than time 0.

2.1.15 IEEE-1588 Hardware Assist

In a distributed control system containing multiple clocks, individual clocks tend to drift apart. Some kind of correction mechanism is necessary to synchronize the individual clocks to maintain global time, which is accurate to some clock resolution. The IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems can be used for this purpose.

The IEEE 1588 standard defines several messages that can be used to exchange timing information. The hardware assist logic required to achieve precision clock synchronization using the IEEE 1588 standard is left to implementation.

The IXP45X/IXP46X network processors consist of this IEEE 1588 hardware-assist logic on three of the MII interfaces. Using the hardware assist logic along with software running on the Intel XScale processor, a full source- or sink-capable IEEE-1588 compliant network node can be implemented.

Note: The Intel® IXP455 Network Processor does not support IEEE-1588 hardware assist. For details, see the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*.

2.1.16 Synchronous Serial Protocol Interface

The IXP45X/IXP46X network processors consist of a dedicated Synchronous Serial Protocol (SSP) interface. The SSP interface is a full-duplex synchronous serial interface. It can connect to a variety of external analog-to-digital (A/D) converters, audio and telecom CODECs, and many other devices which use serial protocols for transferring data.

It supports National's Microwire*, Texas Instruments' synchronous serial protocol (SSP), and Motorola's serial peripheral interface (SPI) protocol.

The SSP operates in master mode (the attached peripheral functions as a slave), and supports serial bit rates from 7.2 Kbps to 1.8432 Mbps using the on-chip, 3.6864-MHz clock, and bit rates from 64.45 Kbps to 16.5 Kbps using a maximum off-chip, 33-MHz clock. Serial data formats may range from 4 to 16 bits in length. Two on-chip register blocks function as independent FIFOs for data, one for each direction. The FIFOs are 16 entries deep x 16 bits wide. Each 32-bit word from the system fills one entry in a FIFO using the lower half 16-bits of a 32-bit word.



2.1.17 I²C Interface

The I²C Bus Interface Unit allows the IXP45X/IXP46X network processors to serve as a master and slave device residing on the I²C bus. The I²C bus is a serial bus consisting of a two-pin interface. SDA is the data pin for input and output functions and SCL is the clock pin for reference and control of the I²C bus.

The I²C bus allows the IXP45X/IXP46X network processors to interface to other I²C peripherals and microcontrollers for system management functions. The serial bus requires a minimum of hardware for an economical system to relay status and reliability information on the IXP45X/IXP46X network processors subsystem to an external device.

The I²C Bus Interface Unit is a peripheral device that resides on the IXP45X/IXP46X network processors' APB. Data is transmitted to and received from the I²C bus via a buffered interface. Control and status information is relayed through a set of memory-mapped registers. Refer to the I²C Bus Specification for complete details on I²C bus operation.

The I²C supports:

- Multi-master capabilities
- Slave capabilities

The I²C unit supports both fast-mode operation — at 400 Kbps — and standard mode — at 100 Kbps. Fast mode logic levels, formats, capacitive loading and protocols function the same in both modes. The I²C unit does not support I²C 10-bit addressing or CBUS.

2.1.18 AES/DES/SHA/MD-5

The IXP45X/IXP46X network processors implement on chip hardware acceleration for underlying security and authentication algorithms.

The encryption/decryption algorithms supported are AES, single pass AES-CCM, triple DES. These algorithms are commonly found when implementing IPsec, VPN, WEP, WEP2, WPA, and WPA2.

The authentication algorithms supported are MD-5, SHA-1, SHA-256, SHA-384, and SHA-512. Inclusion of SHA-384 and SHA-512 allows 256-bit key authentication to pair up with 256-bit AES support.

Note: All the described NPE functions require Intel-supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.19 Cryptography Unit

The Cryptography Unit consists of three major functions:

- Exponentiation Unit (EAU)
- Random Number Generator (RNG)
- Secure Hash Algorithm Unit (SHA)

The EAU supports various large number arithmetic operations. These operations include modular exponentiation, modular reduction, multiply, add and subtract. These operations are controlled through a set of memory mapped registers. Parameters for and results of the operations are written in little-endian ordering into a RAM (contained



within the EAU) which the EAU state machine accesses and also uses for temporary registers. The arithmetic operations supported by the EAU are used by software executing in the host processor to build larger cryptographic functions such as signing and verification procedures. Since the EAU executes only one operation at a time, the host processor must serialize the required operations to the EAU.

The EAU begins operating after the host processor has moved data into the EAU RAM and loads the EAU's command register with an appropriate command. After executing the command, the EAU appropriately sets its status bits and waits idle until it receives another command from the host processor.

The RNG unit provides a digital random number generation capability. It uses a LFSR (Linear Feedback Shift Register) to generate a sequence of pseudo-random bits. These sequences are shifted into a FIFO of 32-bit words, which may be read sequentially from the random number register. A new word is generated every 32 clocks, and the RNG will buffer 16 of these words at a time. The output of the RNG should be passed through the SHA engine for added randomness. The host processor (Intel XScale processor) is responsible for implementing this SHA based random number generation. The LFSR also allows one entropy source. The entropy source is fed in from a PN sequence generator which has a period of $2^{42} - 1$. The coefficients for the PN sequence is chosen such that it produces the maximal sequence length. The coefficients are not mentioned for security reasons. The coefficients for the 128 stage LFSR are similarly not mentioned here for security reasons.

The SHA unit is used to provide added randomness to the random numbers produced by the random number generator (RNG). It is the responsibility of the Intel XScale processor to get the data from the RNG, provide it to the SHA unit, and retrieve the data after it is complete.

2.1.20 Queue Manager

The Queue Manager provides a means for maintaining coherency for data handling between various processor cores contained on the IXP45X/IXP46X network processors (NPE to NPE, NPE to Intel XScale processor, etc.). It maintains the queues as circular buffers in an embedded 8-Kbyte SRAM. The Queue Manager also implements the status flags and pointers required for each queue.

The Queue Manager manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The Queue Manager interfaces include an Advanced High-performance Bus (AHB) interface to the NPEs and Intel XScale processor (or any other AHB bus master), a Flag Bus interface, an event bus (to the NPE condition select logic), and two interrupts to the Intel XScale processor.

The AHB interface is used for configuration of the Queue Manager and provides access to queues, queue status, and SRAM. Individual queue status for queues 0-31 is communicated to the NPEs via the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs via the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

2.2 Intel XScale® Processor

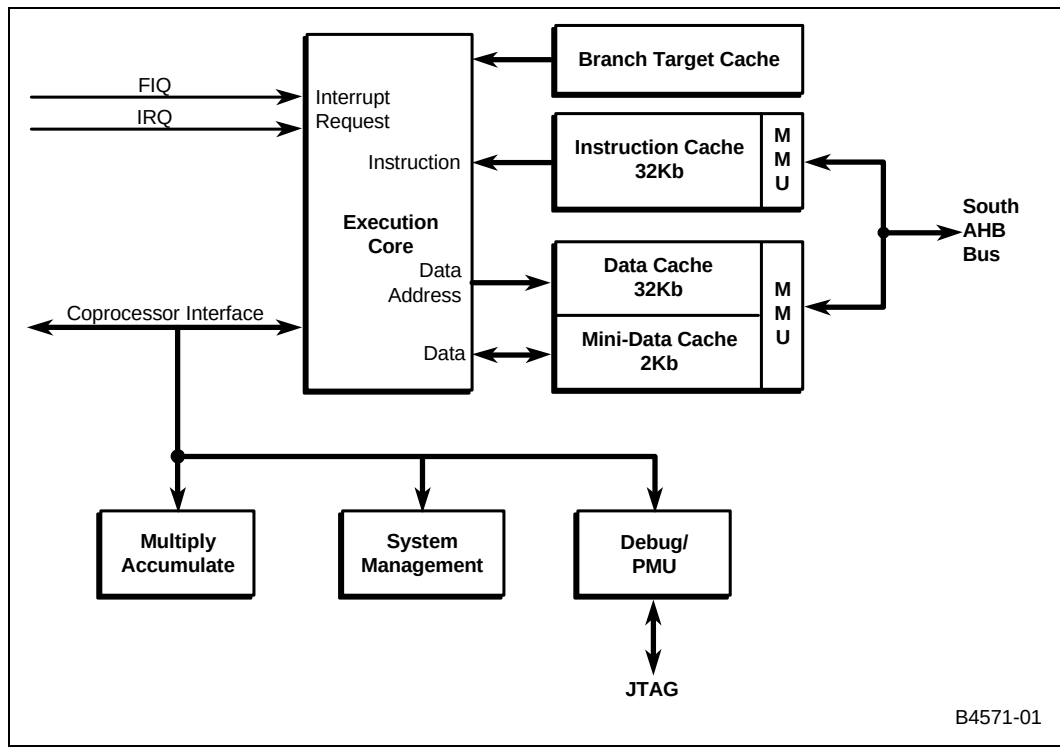
The Intel XScale technology is compliant with the Intel® StrongARM® Version 5TE instruction-set architecture (ISA). The Intel XScale processor — shown in [Figure 4](#) — is designed with Intel, 0.18-μ production semiconductor process technology. This process technology — with the compactness of the Intel® StrongARM® RISC ISA — enables the Intel XScale processor to operate over a wide speed and power range, producing industry-leading mW/MIPS performance.



Intel XScale processor features include:

- Seven/eight-stage super-pipeline promotes high-speed, efficient performance
- 128-entry branch target buffer keeps pipeline filled with statistically correct branch choices
- 32-entry instruction memory-management unit for logical-to-physical address translation, access permissions, and Instruction-Cache (I-cache) attributes
- 32-entry data-memory management unit for logical-to-physical address translation, access permissions, Data-Cache (D-Cache) attributes
- 32-Kbyte instruction cache can hold entire programs, preventing stalls caused by multi-cycle memory accesses
- 32-Kbyte data cache reduces stalls caused by multi-cycle memory accesses
- 2-Kbyte mini-data cache for frequently changing data streams avoids “thrashing” of the D-cache
- Four-entry, fill-and-pend buffers to promote efficiency by allowing “hit-under-miss” operation with data caches
- Eight-entry write buffer allows the Intel XScale processor to continue execution while data is written to memory
- Multiple-accumulate coprocessor that can do two simultaneous, 16-bit, SIMD multiplies with 40-bit accumulation for efficient, high-quality media and signal processing
- Performance monitoring unit (PMU) furnishing two 32-bit event counters and one 32-bit cycle counter for analysis of hit rates, etc.
This PMU is for the Intel XScale processor only. An additional PMU is supplied for monitoring of internal bus performance.
- JTAG debug unit that uses hardware break points and 256-entry trace history buffer (for flow-change messages) to debug programs
- Supports big-endian and little-endian byte ordering

Figure 4. Intel XScale® Processor Block Diagram



2.2.1 Super Pipeline

The super pipeline is composed of integer, multiply-accumulate (MAC), and memory pipes.

The integer pipe has seven stages:

- Branch Target Buffer (BTB)/Fetch 1
- Fetch 2
- Decode
- Register File/Shift
- ALU Execute
- State Execute
- Integer Writeback

The memory pipe has eight stages:

- The first five stages of the Integer pipe (BTB/Fetch 1 through ALU Execute) . . . then finish with the following memory stages
- Data Cache 1
- Data Cache 2
- Data Cache Writeback

The MAC pipe has six to nine stages:



- The first four stages of the Integer pipe (BTB/Fetch 1 through Register File/ Shift) . . . then finish with the following MAC stages
- MAC 1
- MAC 2
- MAC 3
- MAC 4
- Data Cache Writeback

The MAC pipe supports a data-dependent early terminate where stages MAC 2, MAC 3, and/or MAC 4 are bypassed.

Deep pipes promote high instruction execution rates only when a means exists to successfully predict the outcome of branch instructions. The branch target buffer provides such a means.

2.2.2 Branch Target Buffer

Each entry of the 128-entry Branch Target Buffer (BTB) contains the address of a branch instruction, the target address associated with the branch instruction, and a previous history of the branch being taken or not taken. The history is recorded as one of four states:

- Strongly taken
- Weakly taken
- Weakly not taken
- Strongly not taken

The BTB can be enabled or disabled via Coprocessor 15, Register 1.

When the address of the branch instruction hits in the BTB and its history is strongly or weakly taken, the instruction at the branch target address is fetched. When its history is strongly or weakly not-taken, the next sequential instruction is fetched. In either case the history is updated.

Data associated with a branch instruction enters the BTB the first time the branch is taken. This data enters the BTB in a slot with a history of strongly not-taken (overwriting previous data when present).

Successfully predicted branches avoid any branch-latency penalties in the super pipeline. Unsuccessfully predicted branches result in a four-to-five-cycle, branch-latency penalty in the super pipeline.

2.2.3 Instruction Memory Management Unit

For instruction pre-fetches, the Instruction Memory Management Unit (IMMU) controls logical-to-physical address translation, memory access permissions, memory-domain identifications, and attributes (governing operation of the instruction cache).

The IMMU contains a 32-entry, fully associative instruction-translation, look-aside buffer (ITLB) that has a round-robin replacement policy. ITLB entries zero through 30 can be locked.

When an instruction pre-fetch misses in the ITLB, the IMMU invokes an automatic table-walk mechanism that fetches an associated descriptor from memory and loads it into the ITLB. The descriptor contains information for logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes governing operation of the I-cache. The IMMU then continues the instruction



pre-fetch by using the address translation just entered into the ITLB. When an instruction pre-fetch hits in the ITLB, the IMMU continues the pre-fetch using the address translation already resident in the ITLB.

Access permissions for each of up to 16 memory domains can be programmed. When an instruction pre-fetch is attempted to an area of memory in violation of access permissions, the attempt is aborted and a pre-fetch abort is sent to the Intel XScale processor for exception processing. The IMMU and DMMU can be enabled or disabled together.

2.2.4 Data Memory Management Unit

For data fetches, the Data Memory Management Unit (DMMU) controls logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes (governing operation of the data cache or mini-data cache and write buffer). The DMMU contains a 32-entry, fully associative data-translation, look-aside buffer (DTLB) that has a round-robin replacement policy. DTLB entries 0 through 30 can be locked.

When a data fetch misses in the DTLB, the DMMU invokes an automatic table-walk mechanism that fetches an associated descriptor from memory and loads it into the DTLB. The descriptor contains information for logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes (governing operation of the D-cache or mini-data cache and write buffer).

The DMMU continues the data fetch by using the address translation just entered into the DTLB. When a data fetch hits in the DTLB, the DMMU continues the fetch using the address translation already resident in the DTLB.

Access permissions for each of up to 16 memory domains can be programmed. When a data fetch is attempted to an area of memory in violation of access permissions, the attempt is aborted and a data abort is sent to the Intel XScale processor for exception processing.

The IMMU and DMMU can be enabled or disabled together.

2.2.5 Instruction Cache

The Instruction Cache (I-Cache) can contain high-use, multiple-code segments or entire programs, allowing the Intel XScale processor access to instructions at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory.

The 32-Kbyte I-cache is 32-set/32-way associative, where each set contains 32 ways and each way contains a tag address, a cache line of instructions (eight 32-bit words and one parity bit per word), and a line-valid bit. For each of the 32 sets, 0 through 28 ways can be locked. Unlocked ways are replaceable via a round-robin policy.

The I-cache can be enabled or disabled. Attribute bits within the descriptors — contained in the ITLB of the IMMU — provide some control over an enabled I-cache.

When a needed line (eight 32-bit words) is not present in the I-cache, the line is fetched (critical word first) from memory via a two-level, deep-fetch queue. The fetch queue allows the next instruction to be accessed from the I-cache, but only when its data operands do not depend on the execution results of the instruction being fetched via the queue.



2.2.6 Data Cache

The Data Cache (D-Cache) can contain high-use data such as lookup tables and filter coefficients, allowing the Intel XScale processor access to data at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory.

The 32-Kbyte D-cache is 32-set/32-way associative, where each set contains 32 ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two eight-byte groupings in a line), and one valid bit. For each of the 32 sets, zero through 28 ways can be locked, unlocked, or used as local SRAM. Unlocked ways are replaceable via a round-robin policy.

The D-cache (together with the mini-data cache) can be enabled or disabled. Attribute bits within the descriptors, contained in the DTLB of the DMMU, provide significant control over an enabled D-cache. These bits specify cache operating modes such as read and write allocate, write-back, write-through, and D-cache versus mini-data cache targeting.

The D-cache (and mini-data cache) work with the load buffer and pend buffer to provide “hit-under-miss” capability that allows the Intel XScale processor to access other data in the cache after a “miss” is encountered. The D-cache (and mini-data cache) works in conjunction with the write buffer for data that is to be stored to memory.

2.2.7 Mini-Data Cache

The mini-data cache can contain frequently changing data streams such as MPEG video, allowing the Intel XScale processor access to data streams at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory. The mini-data cache relieves the D-cache of data “thrashing” caused by frequently changing data streams.

The 2-Kbyte, mini-data cache is 32-set/two-way associative, where each set contains two ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two eight-byte groupings in a line), and a valid bit. The mini-data cache uses a round-robin replacement policy, and cannot be locked.

The mini-data cache (together with the D-cache) can be enabled or disabled. Attribute bits contained within a coprocessor register specify operating modes write and/or read allocate, write-back, and write-through.

The mini-data cache (and D-cache) work with the load buffer and pend buffer to provide “hit-under-miss” capability that allows the Intel XScale processor to access other data in the cache after a “miss” is encountered. The mini-data cache (and D-cache) works in conjunction with the write buffer for data that is to be stored to memory.

2.2.8 Fill Buffer and Pend Buffer

The four-entry fill buffer (FB) works with the Intel XScale processor to hold non-cacheable loads until the bus controller can act on them. The FB and the four-entry pend buffer (PB) work with the D-cache and mini-data cache to provide “hit-under-miss” capability, allowing the Intel XScale processor to seek other data in the caches while “miss” data is being fetched from memory.

The FB can contain up to four unique “miss” addresses (logical), allowing four “misses” before the Intel XScale processor is stalled. The PB holds up to four addresses (logical) for additional “misses” to those addresses that are already in the FB. A coprocessor register can specify draining of the fill and pend (write) buffers.



2.2.9 Write Buffer

The write buffer (WB) holds data for storage to memory until the bus controller can act on it. The WB is eight entries deep, where each entry holds 16 bytes. The WB is constantly enabled and accepts data from the Intel XScale processor, D-cache, or mini-data cache.

Coprocessor 15, Register 1 specifies whether WB coalescing is enabled or disabled. When coalescing is disabled, stores to memory occur in program order regardless of the attribute bits within the descriptors located in the DTLB.

When coalescing is enabled, the attribute bits within the descriptors located in the DTLB are examined to determine when coalescing is enabled for the destination region of memory. When coalescing is enabled in both CP15, R1 and the DTLB, data entering the WB can coalesce with any of the eight entries (16 bytes) and be stored to the destination memory region, but possibly out of program order.

Stores to a memory region specified to be non-cacheable and non-bufferable by the attribute bits within the descriptors located in the DTLB causes the Intel XScale processor to stall until the store completes. A coprocessor register can specify draining of the write buffer.

2.2.10 Multiply-Accumulate Coprocessor

For efficient processing of high-quality, media-and-signal-processing algorithms, the Multiply-Accumulate Coprocessor (CP0) provides 40-bit accumulation of 16 x 16, dual-16 x 16 (SIMD), and 32 x 32 signed multiplies. Special MAR and MRA instructions are implemented to move the 40-bit accumulator to two Intel XScale processor-general registers (MAR) and move two Intel XScale processor-general registers to the 40-bit accumulator (MRA). The 40-bit accumulator can be stored or loaded to or from D-cache, mini-data cache, or memory using two STC or LDC instructions.

The 16 x 16 signed multiply-accumulates (MIAxy) multiply either the high/high, low/low, high/low, or low/high 16 bits of a 32-bit core general register (multiplier) and another 32-bit core general register (multiplicand) to produce a full, 32-bit product that is sign-extended to 40 bits and added to the 40-bit accumulator.

Dual-signed, 16 x 16 (SIMD) multiply-accumulates (MIAPH) multiply the high/high and low/low 16-bits of a packed 32-bit, core-general register (multiplier) and another packed 32-bit, core-general register (multiplicand) to produce two 16-bits products that are both sign-extended to 40 bits and added to the 40-bit accumulator.

The 32 x 32 signed multiply-accumulates (MIA) multiply a 32-bit, core-general register (multiplier) and another 32-bit, core-general register (multiplicand) to produce a 64-bit product where the 40 LSBs are added to the 40-bit accumulator. The 16 x 32 versions of the 32 x 32 multiply-accumulate instructions complete in a single cycle.

2.2.11 Performance Monitoring Unit

The performance monitoring unit (PMU) contains four 32-bit, event counters and one 32-bit, clock counter. The event counters can be programmed to monitor I-cache hit rate, data caches hit rate, ITLB hit rate, DTLB hit rate, pipeline stalls, BTB prediction hit rate, and instruction execution count.



2.2.12 Debug Unit

The debug unit is accessed through the JTAG port. The industry-standard, IEEE 1149.1 JTAG port consists of a test access port (TAP) controller, boundary-scan register, instruction and data registers, and dedicated signals JTG_TMS, JTG_TDI, JTG_TDO, JTG_TCK, and JTG_TRST_N.

The debug unit — when used with debugger application code running on a host system outside of the Intel XScale processor — allows a program, running on the Intel XScale processor, to be debugged. It allows the debugger application code or a debug exception to stop program execution and redirect execution to a debug-handling routine.

Debug exceptions are instruction breakpoint, data breakpoint, software breakpoint, external debug breakpoint, exception vector trap, and trace buffer full breakpoint. Once execution has stopped, the debugger application code can examine or modify the Intel XScale processor's state, coprocessor state, or memory. The debugger application code can then restart program execution.

The debug unit has two hardware-instruction, break point registers; two hardware, data-breakpoint registers; and a hardware, data-breakpoint control register. The second data-breakpoint register can be alternatively used as a mask register for the first data-breakpoint register.

A 256-entry trace buffer provides the ability to capture control flow messages or addresses. A JTAG instruction (LDIC) can be used to download a debug handler via the JTAG port to the mini-instruction cache (the I-cache has a 2-Kbyte, mini-instruction cache, like the mini-data cache, that is used only to hold a debug handler).





3.0 Intel XScale® Processor

This chapter provides functional descriptions of the Intel XScale® Processor.

3.1 Memory Management Unit

This section describes the memory management unit implemented in the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors.

The Intel XScale® Processor implements the Memory Management Unit (MMU) Architecture specified in the *ARM* Architecture Reference Manual*. To accelerate virtual-to-physical address translation, Intel XScale processor uses both an instruction Translation Look-Aside Buffer (TLB) and a data TLB to cache the latest translations. Each TLB holds 32 entries and is fully associative.

Not only do the TLBs contain the translated addresses, but also the access rights for memory references.

If an instruction or data TLB miss occurs, a hardware translation-table-walking mechanism is invoked to translate the virtual address to a physical address. Once translated, the physical address is placed in the TLB along with the access rights and attributes of the page or section. These translations can also be locked down in either TLB to guarantee the performance of critical routines.

For more information, refer to [“Exceptions” on page 72](#).

The Intel XScale processor allows system software to associate various attributes with regions of memory:

- Cacheable
- Bufferable
- Line-allocate policy
- Write policy
- I/O
- Mini data cache
- Coalescing

For a description of page attributes, see [“Cacheable \(C\), Bufferable \(B\), and eXtension \(X\) Bits” on page 70](#). For information on where these attributes have been mapped in the MMU descriptors, see [“New Page Attributes” on page 175](#).

Note:

The virtual address with which the TLBs are accessed may be remapped by the PID register. For a description of the PID register, see [“Register 13: Process ID” on page 106](#).



Intel® StrongARM® MMU Version 5 Architecture introduces the support of tiny pages, which are 1 KByte in size. The reserved field in the first-level descriptor (encoding 0b11) is used as the fine page table base address. The exact bit fields and the format of the first and second-level descriptors can be found in [“New Page Attributes” on page 175](#).

The attributes associated with a particular region of memory are configured in the memory management page table and control the behavior of accesses to the instruction cache, data cache, mini-data cache, and the write buffer. These attributes are ignored when the MMU is disabled.

To allow compatibility with older system software, the new Intel XScale processor attributes take advantage of encoding space in the descriptors that was formerly reserved.

3.1.1 Memory Attributes

3.1.1.1 Page (P) Attribute Bit

The selection between address or data coherency is controlled by a software-programmable P-Attribute bit in the Intel XScale processor's Memory Management Unit (MMU) and the BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit will be from Expansion Bus Controller Configuration Register 1, bit 8. This bit will reset to 0.

The default endian-conversion method for the IXP45X/IXP46X network processors is address coherency. This was selected for backward compatibility with the Intel® IXP425 Network Processor.

The BYTE_SWAP_EN bit is an enable bit that allows data coherency to be performed, based on the P-Attribute bit.

- When the bit is 0, address coherency is always performed.
- When the bit is 1, the type of coherency performed is dependent on the P-Attribute bit.

The P-Attribute bit is associated with each 1-Mbyte page. The P-Attribute bit is output from the Intel XScale processor with any store or load access associated with that page.

Note: The P-attribute feature allows software to control byte swapping per 1-Mbyte regions. The P-attribute bit selectively enables or disables this byte swap feature. Future Intel XScale processor products with L2 push cache will not support NPE byte-swapping per 1-Mbyte regions. Using the P-attribute bit to byte swap all of the NPE memory region will allow compatible software code porting to future Intel XScale® Processor. Using the P-attribute bit to byte-swap 1-Mbyte regions of the NPE memory will *not* allow compatible software code porting to future Intel XScale technology.

3.1.1.2 Cacheable (C), Bufferable (B), and eXtension (X) Bits

3.1.1.2.1 Instruction Cache

When examining these bits in a descriptor, the Instruction Cache only utilizes the C bit. If the C bit is clear, the Instruction Cache considers a code fetch from that memory to be non-cacheable and will not fill a cache entry. If the C bit is set, then fetches from the associated memory region will be cached.



3.1.1.2.2 Details on Data Cache and Write Buffer Behavior

If the MMU is disabled, all data accesses will be non-cacheable and non-bufferable. This is the same behavior as when the MMU is enabled and a data access uses a descriptor with X, C, and B all set to 0.

The X, C, and B bits determine when the processor should place new data into the Data Cache. The cache places data into the cache in lines (also called blocks). Thus, the basis for making a decision about placing new data into the cache is called a “Line-Allocation Policy.”

If the Line-Allocation Policy is read-allocate, all load operations that miss the cache, request a 32-byte cache line from external memory and allocate it into either the data cache or mini-data cache. (This statement assumes that the cache is enabled.) Store operations that miss the cache will not cause a line to be allocated.

If read/write-allocate is in effect, load **or** store operations that miss the cache will request a 32-byte cache line from external memory if the cache is enabled.

The other policy determined by the X, C, and B bits is the Write Policy. A write-through policy instructs the data cache to keep external memory coherent by performing stores to both external memory and the cache. A write-back policy only updates external memory when a line in the cache is cleaned or needs to be replaced with a new line. Generally, write-back provides higher performance because it generates less data traffic to external memory. For more details on cache policies, see “[Cacheability](#)” on [page 88](#)

3.1.1.2.3 Data Cache and Write Buffer

All of these descriptor bits affect the behavior of the Data Cache and the Write Buffer.

If the X bit for a descriptor is zero, the C and B bits operate as mandated by the Intel® StrongARM® architecture, refer to the *ARM® Architecture Reference Manual*. This behavior is detailed in [Table 5](#).

If the X bit for a descriptor is one, the C and B bits’ meaning is extended, as detailed in [Table 6](#).

Table 5. Data Cache and Buffer Behavior When X = 0

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	N	N	-	-	Stall until complete [†]
0 1	N	Y	-	-	
1 0	Y	Y	Write Through	Read Allocate	
1 1	Y	Y	Write Back	Read Allocate	
[†] Normally, the processor will continue executing after a data access if no dependency on that access is encountered. With this setting, the processor will stall execution until the data access completes. This guarantees to software that the data access has taken effect by the time execution of the data access instruction completes. External data aborts from such accesses will be imprecise (but see “ Data Aborts ” on page 178 for a method to shield code from this imprecision).					

**Table 6. Data Cache and Buffer Behavior When X = 1**

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	-	-	-	-	Unpredictable -- do not use
0 1	N	Y	-	-	Writes will not coalesce into buffers ¹
1 0	(Mini Data Cache)	-	-	-	Cache policy is determined by MD field of Auxiliary Control register ²
1 1	Y	Y	Write Back	Read/Write Allocate	
Notes: 1. Normally, bufferable writes can coalesce with previously buffered data in the same address range. 2. See "Register 1: Control and Auxiliary Control Registers" on page 100 for a description of this register.					

3.1.1.2.4 Memory Operation Ordering

A *fence* memory operation (memop) is one that guarantees all memops issued prior to the fence will execute before any memop issued after the fence. Thus software may issue a fence to impose a partial ordering on memory accesses.

Table 7 on page 72 shows the circumstances in which memops act as fences.

Any swap (**SWP** or **SWPB**) to a page that would create a fence on a load or store is a fence.

Table 7. Memory Operations that Impose a Fence

Operation	X	C	B
load	-	0	-
store	1	0	1
load or store	0	0	0

3.1.1.2.5 Exceptions

The MMU may generate prefetch aborts for instruction accesses and data aborts for data memory accesses. The types and priorities of these exceptions are described in "Event Architecture" on page 177.

Data address alignment checking is enabled by setting bit 1 of the Control Register (CP15, register 1). Alignment faults are still reported even if the MMU is disabled. All other MMU exceptions are disabled when the MMU is disabled.

3.1.2 Interaction of the MMU, Instruction Cache, and Data Cache

The MMU, instruction cache, and data/mini-data cache may be enabled/disabled independently. The instruction cache can be enabled with the MMU enabled or disabled. However, the data cache can only be enabled when the MMU is enabled. Therefore only three of the four combinations of the MMU and data/mini-data cache enables are valid. The invalid combination will cause undefined results.

**Table 8. Valid MMU & Data / Mini-Data Cache Combinations**

MMU	Data / mini-data Cache
Off	Off
On	Off
On	On

3.1.3 MMU Control

3.1.3.1 Invalidate (Flush) Operation

The entire instruction and data TLB can be invalidated at the same time with one command or they can be invalidated separately. An individual entry in the data or instruction TLB can also be invalidated. See [Table 21, “TLB Functions” on page 105](#) for a listing of commands supported by the Intel XScale processor.

Globally invalidating a TLB will not affect locked TLB entries. However, the invalidate-entry operations can invalidate individual locked entries. In this case, the locked contents remain in the TLB, but will never “hit” on an address translation. Effectively, creating a hole in the TLB. This situation may be rectified by unlocking the TLB.

3.1.3.2 Enabling / Disabling

The MMU is enabled by setting bit 0 in coprocessor 15, register 1 (Control Register).

When the MMU is disabled, accesses to the instruction cache default to cacheable accesses and all accesses to data memory are made non-cacheable.

A recommended code sequence for enabling the MMU is shown in [Example 1 on page 74](#).



Example 1. Enabling the MMU

```
; This routine provides software with a predictable way of enabling the MMU.
; After the CPWAIT, the MMU is guaranteed to be enabled. Be aware
; that the MMU will be enabled sometime after MCR and before the instruction
; that executes after the CPWAIT.
; Programming Note: This code sequence requires a one-to-one virtual to
; physical address mapping on this code since
; the MMU may be enabled part way through. This would allow the instructions
; after MCR to execute properly regardless the state of the MMU.

MRC P15,0,R0,C1,C0,0; Read CP15, register 1
ORR R0, R0, #0x1; Turn on the MMU
MCR P15,0,R0,C1,C0,0; Write to CP15, register 1

; For a description of CPWAIT, see
; "Additions to CP15 Functionality" on page 176
CPWAIT
; The MMU is guaranteed to be enabled at this point; the next instruction or
; data address will be translated.
```

3.1.3.3 Locking Entries

Individual entries can be locked into the instruction and data TLBs. See [Table 22, "Cache Lock-Down Functions" on page 105](#) for the exact commands. If a lock operation finds the virtual address translation already resident in the TLB, the results are unpredictable. An invalidate by entry command before the lock command will ensure proper operation. Software can also accomplish this by invalidating all entries, as shown in [Example 2 on page 75](#).

Locking entries into either the instruction TLB or data TLB reduces the available number of entries (by the number that was locked down) for hardware to cache other virtual to physical address translations.

A procedure for locking entries into the instruction TLB is shown in [Example 2 on page 75](#).

If a MMU abort is generated during an instruction or data TLB lock operation, the Fault Status Register is updated to indicate a Lock Abort (see ["Data Aborts" on page 178](#)), and the exception is reported as a data abort.



Example 2. Locking Entries into the Instruction TLB

```

; R1, R2 and R3 contain the virtual addresses to translate and lock into
; the instruction TLB.

; The value in R0 is ignored in the following instruction.
; Hardware guarantees that accesses to CP15 occur in program order

MCR P15,0,R0,C8,C5,0      ; Invalidate the entire instruction TLB

MCR P15,0,R1,C10,C4,0     ; Translate virtual address (R1) and lock into
                           ; instruction TLB

MCR P15,0,R2,C10,C4,0     ; Translate
                           ; virtual address (R2) and lock into instruction TLB

MCR P15,0,R3,C10,C4,0     ; Translate virtual address (R3) and lock into
                           ; instruction TLB

CPWAIT

; The MMU is guaranteed to be updated at this point; the next instruction will
; see the locked instruction TLB entries.

```

Note: If exceptions are allowed to occur in the middle of this routine, the TLB may end up caching a translation that is about to be locked. For example, if R1 is the virtual address of an interrupt service routine and that interrupt occurs immediately after the TLB has been invalidated, the lock operation will be ignored when the interrupt service routine returns back to this code sequence. Software should disable interrupts (FIQ or IRQ) in this case.

As a general rule, software should avoid locking in all other exception types.

The proper procedure for locking entries into the data TLB is shown in [Example 3 on page 76](#).



Example 3. Locking Entries into the Data TLB

```
; R1, and R2 contain the virtual addresses to translate and lock into the data TLB

MCR P15,0,R1,C8,C6,1      ; Invalidate the data TLB entry specified by the
                           ; virtual address in R1
MCR P15,0,R1,C10,C8,0     ; Translate virtual address (R1) and lock into
                           ; data TLB

; Repeat sequence for virtual address in R2
MCR P15,0,R2,C8,C6,1      ; Invalidate the data TLB entry specified by the
                           ; virtual address in R2
MCR P15,0,R2,C10,C8,0     ; Translate virtual address (R2) and lock into
                           ; data TLB

CPWAIT                    ; wait for locks to complete

; The MMU is guaranteed to be updated at this point; the next instruction will
; see the locked data TLB entries.
```

Note: Care must be exercised here when allowing exceptions to occur during this routine whose handlers may have data that lies in a page that is trying to be locked into the TLB.

3.1.3.4 Round-Robin Replacement Algorithm

The line replacement algorithm for the TLBs is round-robin; there is a round-robin pointer that keeps track of the next entry to replace. The next entry to replace is the one sequentially after the last entry that was written. For example, if the last virtual to physical address translation was written into entry 5, the next entry to replace is entry 6.

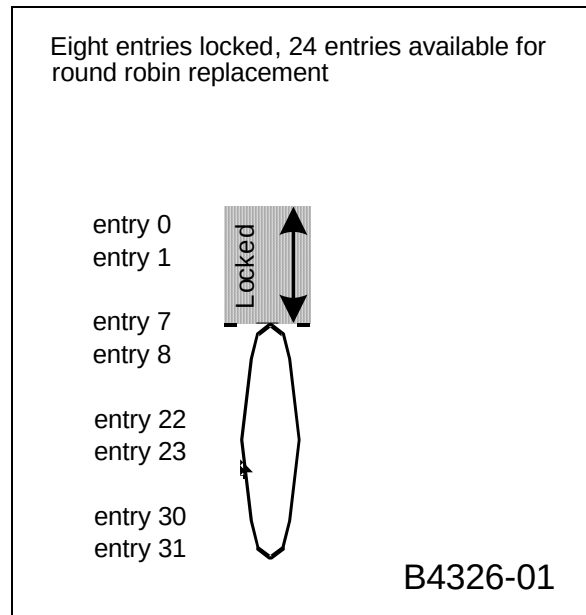
At reset, the round-robin pointer is set to entry 31. Once a translation is written into entry 31, the round-robin pointer gets set to the next available entry, beginning with entry 0 if no entries have been locked down. Subsequent translations move the round-robin pointer to the next sequential entry until entry 31 is reached, where it will wrap back to entry 0 upon the next translation.

A lock pointer is used for locking entries into the TLB and is set to entry 0 at reset. A TLB lock operation places the specified translation at the entry designated by the lock pointer, moves the lock pointer to the next sequential entry, and resets the round-robin pointer to entry 31. Locking entries into either TLB effectively reduces the available entries for updating. For example, if the first three entries were locked down, the round-robin pointer would be entry 3 after it rolled over from entry 31.



Only entries 0 through 30 can be locked in either TLB; entry 31 can never be locked. If the lock pointer is at entry 31, a lock operation will update the TLB entry with the translation and ignore the lock. In this case, the round-robin pointer will stay at entry 31.

Figure 5. Example of Locked Entries in TLB



3.2 Instruction Cache

The Intel XScale processor instruction cache enhances performance by reducing the number of instruction fetches from external memory. The cache provides fast execution of cached code. Code can also be locked down when guaranteed or fast access time is required.

Figure 6 shows the cache organization and how the instruction address is used to access the cache.

The instruction cache is available as a 32 K, 32-way set, associative cache. Each set is 1,024 bytes in size. Each set contains 32 ways. Each way of a set contains eight 32-bit words and one valid bit, which is referred to as a line. The replacement policy is a round-robin algorithm and the cache also supports the ability to lock code in at a line granularity.

The instruction cache is virtually addressed and virtually tagged.

Note: The virtual address presented to the instruction cache may be remapped by the PID register. For a description of the PID register, see [“Register 13: Process ID” on page 106](#).

3.2.1 Operation When Instruction Cache is Enabled

When the cache is enabled, it compares every instruction request address against the addresses of instructions that it is currently holding. If the cache contains the requested instruction, the access “hits” the cache, and the cache returns the requested instruction. If the cache does not contain the requested instruction, the access “misses” the cache. The cache requests an eight-word, also known as a line, fetch from external

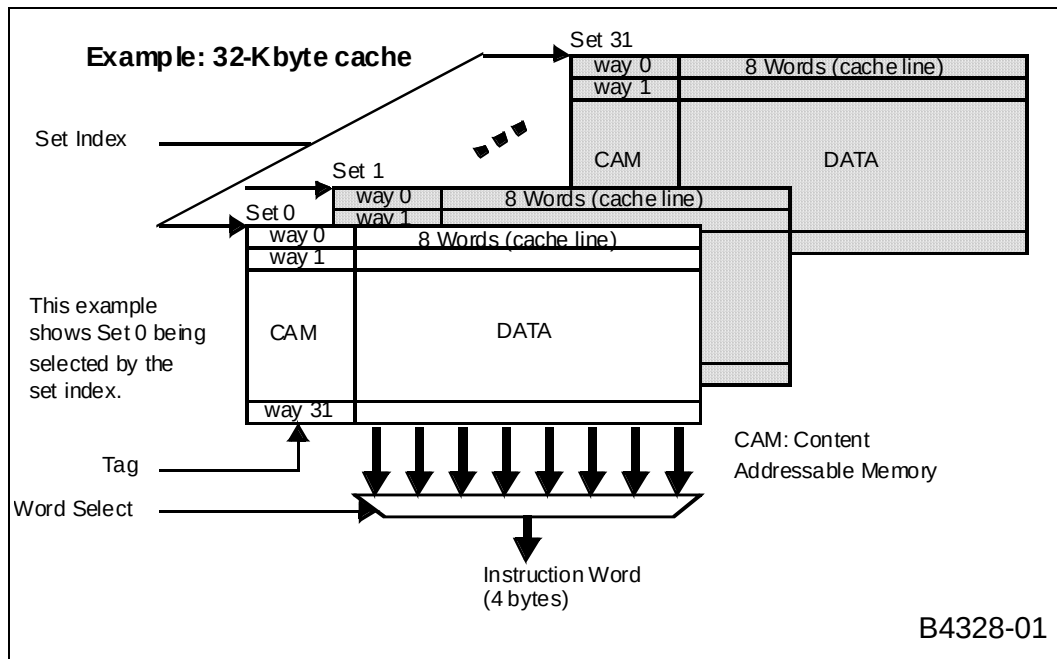


memory that contains the requested instruction using the fetch policy described in [“Instruction-Cache ‘Miss’” on page 78](#). As the fetch returns instructions to the cache, the instructions are placed in one of two fetch buffers and the requested instruction is delivered to the instruction decoder.

A fetched line will be written into the cache if it is cacheable. Code is designated as cacheable when the Memory Management Unit (MMU) is disabled or when the MMU is enable and the cacheable (C) bit is set to 1 in its corresponding page. See [“Memory Management Unit” on page 69](#) for a discussion on page attributes.

Note that an instruction fetch may “miss” the cache but “hit” one of the fetch buffers. When this happens, the requested instruction will be delivered to the instruction decoder in the same manner as a cache “hit.”

Figure 6. Instruction Cache Organization



Disabling the cache prevents any lines from being written into the instruction cache. Although the cache is disabled, it is still accessed and may generate a “hit” if the data is already in the cache.

Disabling the instruction cache **does not** disable instruction buffering that may occur within the instruction fetch buffers. Two 8-word instruction fetch buffers will always be enabled in the cache disabled mode. So long as instruction fetches continue to “hit” within either buffer (even in the presence of forward and backward branches), no external fetches for instructions are generated. A miss causes one or the other buffer to be filled from external memory using the fill policy described in [“Instruction-Cache ‘Miss’” on page 78](#).

3.2.1.1 Instruction-Cache ‘Miss’

An instruction-cache “miss” occurs when the requested instruction is not found in the instruction fetch buffers or instruction cache; a fetch request is then made to external memory. The instruction cache can handle up to two “misses.” Each external fetch request uses a fetch buffer that holds 32-bytes and eight valid bits, one for each word.



A miss causes the following:

- A fetch buffer is allocated
 - The instruction cache sends a fetch request to the external bus. This request is for a 32-byte line.
 - Instructions words are returned back from the external bus, at a maximum rate of 1 word per core cycle. The instruction cache can have the eight words of data return in any order, which allows the Intel XScale processor to send the requested instruction first, thus reducing fetch latency. (This is referred to as critical word first.) As each word returns, the corresponding valid bit is set for the word in the fetch buffer.
 - As soon as the fetch buffer receives the requested instruction, it forwards the instruction to the instruction decoder for execution.
 - When all words have returned, the fetched line will be written into the instruction cache if it is cacheable and if the instruction cache is enabled. The line chosen for update in the cache is controlled by the round-robin replacement algorithm. This update may evict a valid line at that location. For more information on enabling or disabling instruction cache, refer to [“Instruction-Cache Coherence” on page 80](#)
1. Once the cache is updated, the eight valid bits of the fetch buffer are invalidated.

3.2.1.2 Instruction-Cache Line-Replacement Algorithm

The line replacement algorithm for the instruction cache is round-robin. Each set in the instruction cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the one after the last line that was written. For example, if the line for the last external instruction fetch was written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way 0 if no lines have been locked into that particular set. Locking lines into the instruction cache effectively reduces the available lines for cache updating. For example, if the first three lines of a set were locked down, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. For more details on cache locking, see [“Instruction-Cache Coherence” on page 80](#).

The instruction cache is protected by parity to ensure data integrity. Each instruction cache word has 1 parity bit. (The instruction cache tag is NOT parity protected.) When a parity error is detected on an instruction cache access, a prefetch abort exception occurs if the Intel XScale processor attempts to execute the instruction. Before servicing the exception, hardware places a notification of the error in the Fault Status Register (Coprocessor 15, register 5).

A software exception handler can recover from an instruction cache parity error. The parity error can be accomplished by invalidating the instruction cache and the branch target buffer and then returning to the instruction that caused the prefetch abort exception. A simplified code example is shown in [Example 4 on page 80](#). A more complex handler might choose to invalidate the specific line that caused the exception and then invalidate the BTB.



Example 4. Recovering from an Instruction Cache Parity Error

```
; Prefetch abort handler
MCR P15,0,R0,C7,C5,0      ; Invalidate the instruction cache and branch target
                           ; buffer
CPWAIT                     ; wait for effect (see "Additions to CP15 Functionality" on page 176
for a                      ; description of CPWAIT)
SUBS PC,R14,#4             ; Returns to the instruction that generated the
                           ; parity error
; The Instruction Cache is guaranteed to be invalidated at this point
```

If a parity error occurs on an instruction that is locked in the cache, the software exception handler needs to unlock the instruction cache, invalidate the cache and then re-lock the code in before it returns to the faulting instruction.

The instruction cache does not detect modification to program memory by loads, stores or actions of other bus masters. Several situations may require program memory modification, such as uploading code from disk.

The application program is responsible for synchronizing code modification and invalidating the cache. In general, software must ensure that modified code space is not accessed until modification and invalidating are completed.

3.2.1.3 Instruction-Cache Coherence

To achieve cache coherence, instruction cache contents can be invalidated after code modification in external memory is complete.

If the instruction cache is not enabled, or code is being written to a non-cacheable region, software must still invalidate the instruction cache before using the newly-written code. This precaution ensures that state associated with the new code is not buffered elsewhere in the processor, such as the fetch buffers or the BTB.

Naturally, when writing code as data, care must be taken to force it completely out of the processor into external memory before attempting to execute it. If writing into a non-cacheable region, flushing the write buffers is sufficient precaution (see ["Register 7: Cache Functions" on page 103](#) for a description of this operation). If writing to a cacheable region, then the data cache should be submitted to a Clean/Invalidate operation (see ["Cacheability" on page 88](#)) to ensure coherency.

After reset, the instruction cache is always disabled, unlocked, and invalidated (flushed).

The instruction cache is enabled by setting bit 12 in coprocessor 15, register 1 (Control Register). This process is illustrated in [Example 5, Enabling the Instruction Cache](#).



Example 5. Enabling the Instruction Cache

```

; Enable the ICache
MRC P15, 0, R0, C1, C0, 0      ; Get the control register
ORR R0, R0, #0x1000           ; set bit 12 -- the I bit
MCR P15, 0, R0, C1, C0, 0      ; Set the control register

```

The entire instruction cache along with the fetch buffers are invalidated by writing to coprocessor 15, register 7. (See [Table 20, “Cache Functions” on page 104](#) for the exact command.) The invalidate command does not unlock any lines that were locked in the instruction cache nor does it invalidate those locked lines. To invalidate the entire cache including locked lines, the unlock instruction cache command needs to be executed before the invalidate command. The unlock command can also be found in [Table 22, “Cache Lock-Down Functions” on page 105](#).

There is an inherent delay from the execution of the instruction cache invalidate command to where the next instruction will see the result of the invalidate. The following routine can be used to guarantee proper synchronization.

Example 6. Invalidating the Instruction Cache

```

MCR P15,0,R1,C7,C5,0      ; Invalidate the instruction cache and branch
                           ; target buffer

CPWAIT

; The instruction cache is guaranteed to be invalidated at this point; the next
; instruction sees the result of the invalidate command.

```

The Intel XScale processor also supports invalidating an individual line from the instruction cache. See [Table 20, “Cache Functions” on page 104](#) for the exact command.

Software has the ability to lock performance critical routines into the instruction cache. Up to 28 lines in each set can be locked; hardware will ignore the lock command if software is trying to lock all the lines in a particular set (i.e., ways 28-31 can never be locked). When all ways in a particular set are requested to be locked, the instruction cache line will still be allocated into the cache but the lock will be ignored. The round-robin pointer will stay at way 31 for that set.

Cache lines can be locked into the instruction cache by initiating a write to coprocessor 15. (See [Table 22, “Cache Lock-Down Functions” on page 105](#) for the exact command.) Register *Rd* contains the virtual address of the line to be locked into the cache.

There are several requirements for locking down code:

- The routine used to lock lines down in the cache must be placed in non-cacheable memory, which means the MMU is enabled.
As a result: no fetches of cacheable code should occur while locking instructions into the cache.



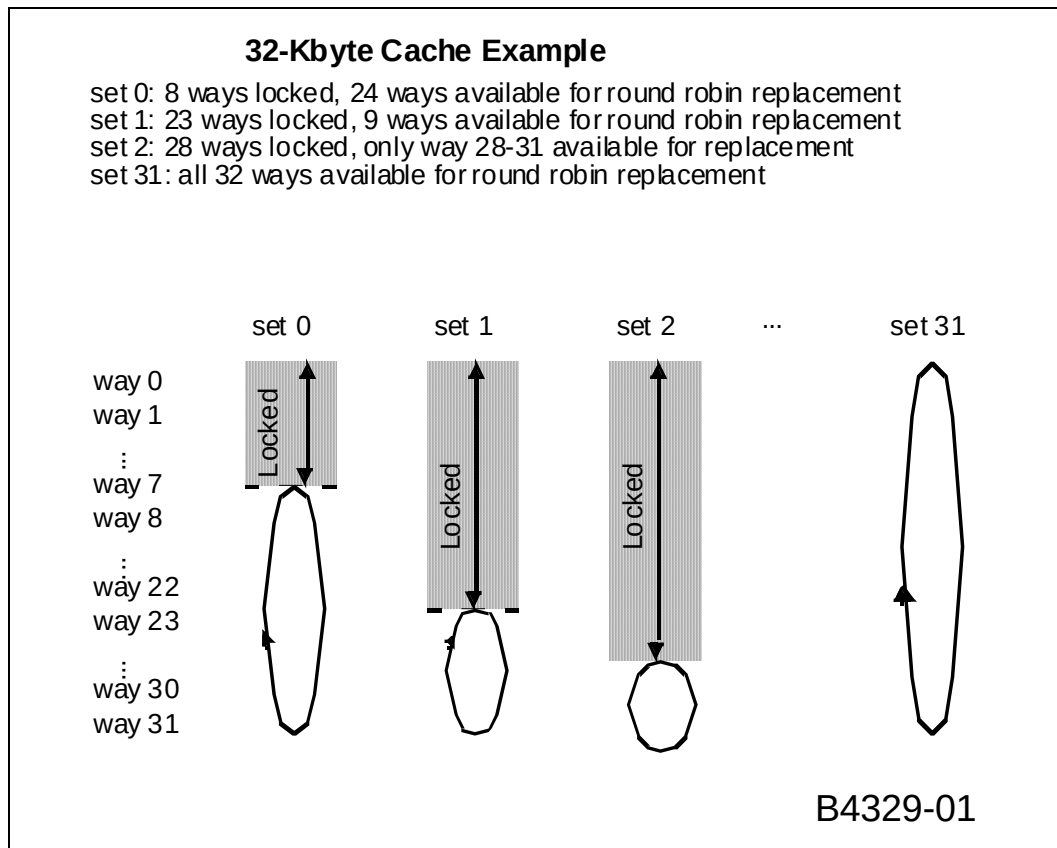
- The code being locked into the cache must be cacheable
- The instruction cache must be enabled and invalidated prior to locking down lines.

Failure to follow these requirements will produce unpredictable results when accessing the instruction cache.

System programmers should ensure that the code to lock instructions into the cache does not reside closer than 128 bytes to a non-cacheable/cacheable page boundary. If the processor fetches ahead into a cacheable page, then the first requirement noted above could be violated.

Lines are locked into a set starting at way 0 and may progress up to way 27; which set a line gets locked into depends on the set index of the virtual address. [Figure 7](#) is an example (32-Kbyte cache) of where lines of code may be locked into the cache along with how the round-robin pointer is affected.

Figure 7. Locked Line Effect on Round-Robin Replacement



Software can lock down several different routines located at different memory locations. This may cause some sets to have more locked lines than others as shown in [Figure 7](#).

[Example 7 on page 83](#) shows how a routine, called “lockMe” in this example, might be locked into the instruction cache. Note that it is possible to receive an exception while locking code (see [“Event Architecture” on page 177](#)).



Example 7. Locking Code into the Cache

```
lockMe:                                ; This is the code that will be locked into the cache

    mov    r0, #5

    add     r5, r1, r2

    . . .

lockMeEnd:

    . . .

codeLock:                                ; here is the code to lock the "lockMe" routine

    ldr     r0, =(lockMe AND NOT 31); r0 gets a pointer to the first line we
    should lock

    ldr     r1, =(lockMeEnd AND NOT 31); r1 contains a pointer to the last line we
    should lock

lockLoop:

    mcr     p15, 0, r0, c9, c1, 0      ; lock next line of code into ICache

    cmp     r0, r1                      ; are we done yet?

    add     r0, r0, #32                 ; advance pointer to next line

    bne     lockLoop                   ; if not done, do the next line
```

The Intel XScale processor provides a global unlock command for the instruction cache. Writing to coprocessor 15, register 9 unlocks all the locked lines in the instruction cache and leaves them valid. These lines then become available for the round-robin replacement algorithm. (See [Table 22, "Cache Lock-Down Functions" on page 105](#) for the exact command.)

3.3 Branch Target Buffer

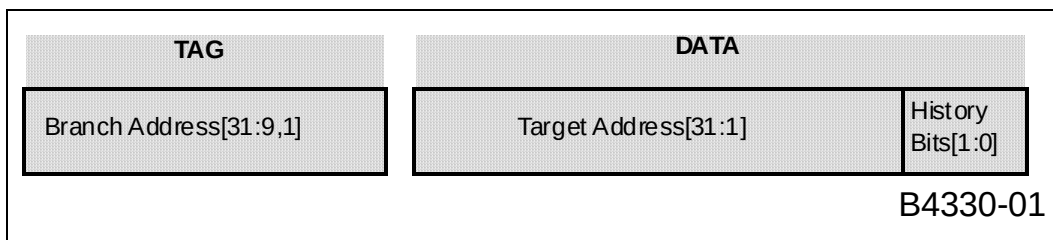
The Intel XScale processor uses dynamic branch prediction to reduce the penalties associated with changing the flow of program execution. The Intel XScale processor features a branch target buffer that provides the instruction cache with the target address of branch type instructions. The branch target buffer is implemented as a 128-entry, direct-mapped cache.

This section is primarily for those optimizing their code for performance. An understanding of the branch target buffer is needed in this case so that code can be scheduled to best utilize the performance benefits of the branch target buffer.

3.3.1 Branch Target Buffer (BTB) Operation

The BTB stores the history of branches that have executed along with their targets. [Figure 8](#) shows an entry in the BTB, where the tag is the instruction address of a previously executed branch and the data contains the target address of the previously executed branch along with two bits of history information.

Figure 8. BTB Entry



The BTB takes the current instruction address and checks to see if this address is a branch that was previously seen. The BTB uses bits [8:2] of the current address to read out the tag and then compares this tag to bits [31:9,1] of the current instruction address. If the current instruction address matches the tag in the cache and the history bits indicate that this branch is usually taken in the past, the BTB uses the data (target address) as the next instruction address to send to the instruction cache.

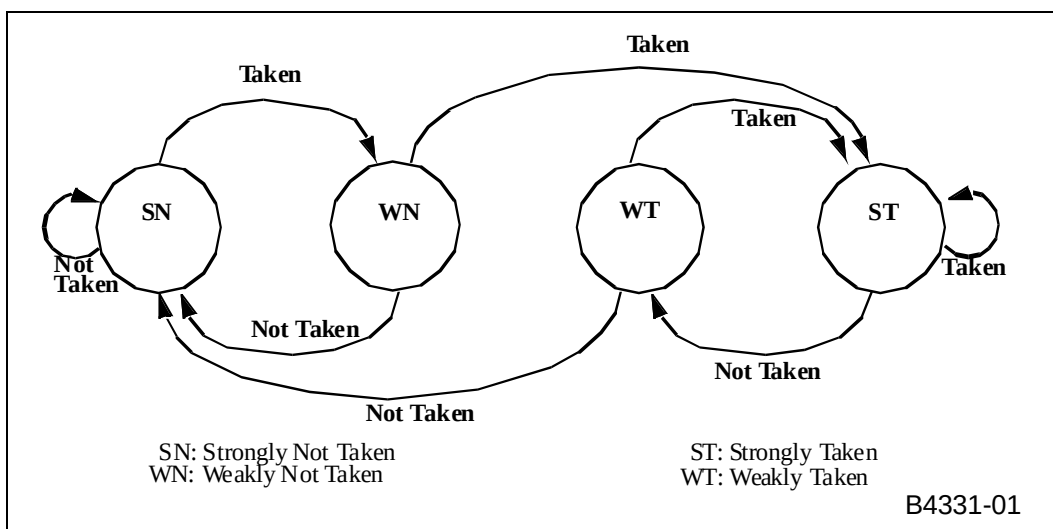
Bit[1] of the instruction address is included in the tag comparison in order to support Thumb execution. This organization means that two consecutive Thumb branch (B) instructions, with instruction address bits[8:2] the same, will contend for the same BTB entry. Thumb also requires 31 bits for the branch target address. In Intel® StrongARM® mode, bit[1] is zero.

The history bits represent four possible prediction states for a branch entry in the BTB. [Figure 9, “Branch History” on page 84](#) shows these states along with the possible transitions. The initial state for branches stored in the BTB is Weakly-Taken (WT). Every time a branch that exists in the BTB is executed, the history bits are updated to reflect the latest outcome of the branch, either taken or not-taken.

[“Performance Considerations” on page 181](#) describes which instructions are dynamically predicted by the BTB and the performance penalty for incorrectly predicting a branch.

The BTB does not have to be managed explicitly by software; it is disabled by default after reset and is invalidated when the instruction cache is invalidated.

Figure 9. Branch History





3.3.1.1 Reset

After Processor Reset, the BTB is disabled and all entries are invalidated.

A new entry is stored into the BTB when the following conditions are met:

- The branch instruction has executed
- The branch was taken
- The branch is not currently in the BTB

The entry is then marked valid and the history bits are set to WT. If another valid branch exists at the same entry in the BTB, it will be evicted by the new branch.

Once a branch is stored in the BTB, the history bits are updated upon every execution of the branch as shown in [Figure 9](#).

The BTB is always disabled with Reset. Software can enable the BTB through a bit in a coprocessor register (see [“Register 1: Control and Auxiliary Control Registers”](#) on [page 100](#)).

Before enabling or disabling the BTB, software must invalidate the BTB (described in the following section). This action will ensure correct operation in case stale data is in the BTB. Software should not place any branch instruction between the code that invalidates the BTB and the code that enables/disables it.

There are four ways the contents of the BTB can be invalidated.

- Reset
- Software can directly invalidate the BTB via a CP15, register 7 function. Refer to [“Register 7: Cache Functions”](#) on [page 103](#).
- The BTB is invalidated when the Process ID Register is written.
- The BTB is invalidated when the instruction cache is invalidated via CP15, register 7 functions.

3.4 Data Cache

The Intel XScale processor data cache enhances performance by reducing the number of data accesses to and from external memory. There are two data cache structures in the Intel XScale processor: a 32-Kbyte data cache and a 2-Kbyte mini-data cache. An eight entry write buffer and a four-entry, fill buffer are also implemented to decouple the Intel XScale processor instruction execution from external memory accesses, which increases overall system performance.

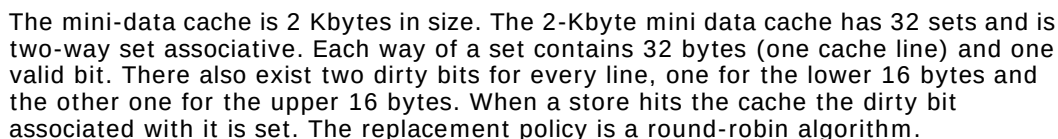
3.4.1 Data Cache Overview

The data cache is a 32-Kbyte, 32-way set, associative cache. The 32-Kbyte cache has 32 sets. Each set contains 32 ways. Each way of a set contains 32 bytes (one cache line) and one valid bit. There also exist two dirty bits for every line, one for the lower 16 bytes and the other one for the upper 16 bytes. When a store hits the cache the dirty bit associated with it is set. The replacement policy is a round-robin algorithm and the cache also supports the ability to reconfigure each line as data RAM.

[Figure 10, “Data Cache Organization”](#) on [page 86](#) shows the cache organization and how the data address is used to access the cache.

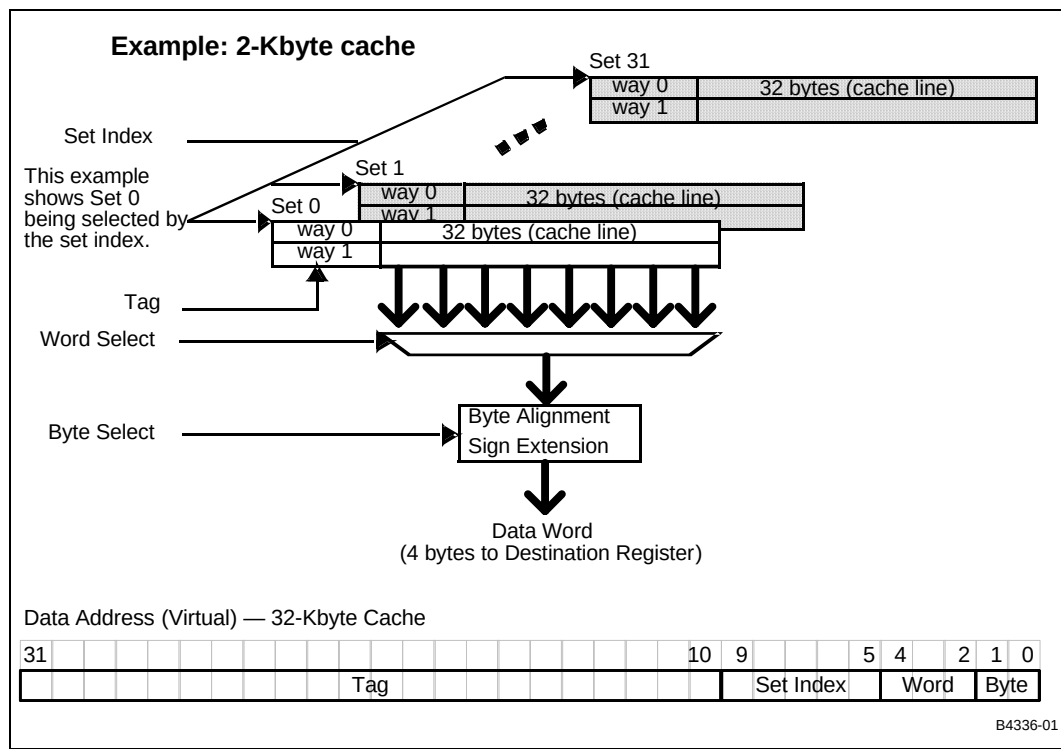
Cache policies may be adjusted for particular regions of memory by altering page attribute bits in the MMU descriptor that controls that memory. See [“Memory Attributes”](#) on [page 70](#) for a description of these bits.

Figure 10. Data Cache Organization



The mini-data cache is virtually addressed and virtually tagged and supports the same caching policies as the data cache. However, lines can't be locked into the mini-data cache.

Figure 11. Mini-Data Cache Organization



The Intel XScale processor employs an eight entry write buffer, each entry containing 16 bytes. Stores to external memory are first placed in the write buffer and subsequently taken out when the bus is available.

The write buffer supports the coalescing of multiple store requests to external memory. An incoming store may coalesce with any of the eight entries.

The fill buffer holds the external memory request information for a data cache or mini-data cache fill or non-cacheable read request. Up to four 32-byte read request operations can be outstanding in the fill buffer before the Intel XScale processor needs to stall.

The fill buffer has been augmented with a four entry pend buffer that captures data memory requests to outstanding fill operations. Each entry in the pend buffer contains enough data storage to hold one 32-bit word, specifically for store operations. Cacheable load or store operations that hit an entry in the fill buffer get placed in the pend buffer and are completed when the associated fill completes. Any entry in the pend buffer can be pending against any of the entries in the fill buffer; multiple entries in the pend buffer can be pending against a single entry in the fill buffer.

Pended operations complete in program order.

The following discussions refer to the data cache and mini-data cache as one cache (data/mini-data) since their behavior is the same when accessed.

When the data/mini-data cache is enabled for an access, the data/mini-data cache compares the address of the request against the addresses of data that it is currently holding. If the line containing the address of the request is resident in the cache, the access “hits” the cache. For a load operation the cache returns the requested data to the destination register and for a store operation the data is stored into the cache. The



data associated with the store may also be written to external memory if write-through caching is specified for that area of memory. If the cache does not contain the requested data, the access ‘misses’ the cache, and the sequence of events that follows depends on the configuration of the cache, the configuration of the MMU and the page attributes, which are described in [“Cacheability” on page 88](#).

The data/mini-data cache is still accessed even though it is disabled. If a load hits the cache it will return the requested data to the destination register. If a store hits the cache, the data is written into the cache. Any access that misses the cache will not allocate a line in the cache when it is disabled, even if the MMU is enabled and the memory region’s cacheability attribute is set.

3.4.2 Cacheability

Data at a specified address is cacheable given the following:

- the MMU is enabled
- the cacheable attribute is set in the descriptor for the accessed address
- and the data/mini-data cache is enabled

The following sequence of events occurs when a cacheable (see [“Cacheability” on page 88](#)) load operation misses the cache:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.
If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it accesses the cache again, to obtain the request data and returns it to the destination register.
If there is no outstanding fill request for that line, the current load request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor will stall until an entry is available.
2. A line is allocated in the cache to receive the 32-bytes of fill data. The line selected is determined by the round-robin pointer. (See [“Cacheability” on page 88](#).) The line chosen may contain a valid line previously allocated in the cache. In this case both dirty bits are examined and if set, the four words associated with a dirty bit that’s asserted will be written back to external memory as a four word burst operation.
3. When the data requested by the load is returned from external memory, it is immediately sent to the destination register specified by the load. A system that returns the requested data back first, with respect to the other bytes of the line, will obtain the best performance.
4. As data returns from external memory it is written into the cache in the previously allocated line.

A load operation that misses the cache and is NOT cacheable makes a request from external memory for the exact data size of the original load request. For example, **LDRH** requests exactly two bytes from external memory, **LDR** requests 4 bytes from external memory, etc. This request is placed in the fill buffer until the data is returned from external memory, which is then forwarded back to the destination register(s).

A write operation that misses the cache will request a 32-byte cache line from external memory if the access is cacheable and write allocation is specified in the page. In this case the following sequence of events occur:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.



If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it writes its data into the recently allocated cache line.

If there is no outstanding fill request for that line, the current store request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor will stall until an entry is available.

2. The 32-bytes of data can be returned back to the Intel XScale processor in any word order, i.e., the eight words in the line can be returned in any order. Note that it does not matter, for performance reasons, which order the data is returned to the Intel XScale processor since the store operation has to wait until the entire line is written into the cache before it can complete.
3. When the entire 32-byte line has returned from external memory, a line is allocated in the cache, selected by the round-robin pointer. The line to be written into the cache may replace a valid line previously allocated in the cache. In this case both dirty bits are examined and if any are set, the four words associated with a dirty bit that's asserted will be written back to external memory as a 4 word burst operation. This write operation will be placed in the write buffer.
4. The line is written into the cache along with the data associated with the store operation.

If the above condition for requesting a 32-byte cache line is not met, a write miss will cause a write request to external memory for the exact data size specified by the store operation, assuming the write request doesn't coalesce with another write operation in the write buffer.

The Intel XScale processor supports write-back caching or write-through caching, controlled through the MMU page attributes. When write-through caching is specified, all store operations are written to external memory even if the access hits the cache. This feature keeps the external memory coherent with the cache, i.e., no dirty bits are set for this region of memory in the data/mini-data cache. However write through does not guarantee that the data/mini-data cache is coherent with external memory, which is dependent on the system level configuration, specifically if the external memory is shared by another master.

When write-back caching is specified, a store operation that hits the cache will not generate a write to external memory, thus reducing external memory traffic.

The line replacement algorithm for the data cache is round-robin. Each set in the data cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the next sequential line after the last one that was just filled. For example, if the line for the last fill was written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way 0 if no lines have been re-configured as data RAM in that particular set. Re-configuring lines as data RAM effectively reduces the available lines for cache updating. For example, if the first three lines of a set were re-configured, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. Refer to ["Reconfiguring the Data Cache as Data RAM" on page 92](#) for more details on data RAM.

The mini-data cache follows the same round-robin replacement algorithm as the data cache except that there are only two lines the round-robin pointer can point to such that the round-robin pointer always points to the least recently filled line. A least recently used replacement algorithm is not supported because the purpose of the mini-



data cache is to cache data that exhibits low temporal locality, i.e., data that is placed into the mini-data cache is typically modified once and then written back out to external memory.

The data cache and mini-data cache are protected by parity to ensure data integrity; there is one parity bit per byte of data. (The tags are NOT parity protected.) When a parity error is detected on a data/mini-data cache access, a data abort exception occurs. Before servicing the exception, hardware will set bit 10 of the Fault Status Register register.

A data/mini-data cache parity error is an imprecise data abort, meaning R14_ABORT (+8) may not point to the instruction that caused the parity error. If the parity error occurred during a load, the targeted register may be updated with incorrect data.

A data abort due to a data/mini-data cache parity error may not be recoverable if the data address that caused the abort occurred on a line in the cache that has a write-back caching policy. Prior updates to this line may be lost; in this case the software exception handler should perform a “clean and clear” operation on the data cache, ignoring subsequent parity errors, and restart the offending process.

The **SWP** and **SWPB** instructions generate an atomic load and store operation allowing a memory semaphore to be loaded and altered without interruption. These accesses may hit or miss the data/mini-data cache depending on configuration of the cache, configuration of the MMU, and the page attributes.

After processor reset, both the data cache and mini-data cache are disabled, all valid bits are set to zero (invalid), and the round-robin bit points to way 31. Any lines in the data cache that were configured as data RAM before reset are changed back to cacheable lines after reset, i.e., there are 32 KBytes of data cache and zero bytes of data RAM.

The data cache and mini-data cache are enabled by setting bit 2 in coprocessor 15, register 1 (Control Register). See “[Configuration](#)” on [page 96](#), for a description of this register and others.

[Example 8](#) shows code that enables the data and mini-data caches. Note that the MMU must be enabled to use the data cache.

Example 8. Enabling the Data Cache

```
enabledDCache:

    MCR    p15, 0, r0, c7, c10, 4; Drain pending data operations...
           ; (see Chapter 7.2.8, Register 7: Cache functions)

    MRC    p15, 0, r0, c1, c0, 0; Get current control register

    ORR    r0, r0, #4          ; Enable DCache by setting 'C' (bit 2)

    MCR    p15, 0, r0, c1, c0, 0; And update the Control register
```

Individual entries can be invalidated and cleaned in the data cache and mini-data cache via coprocessor 15, register 7. Note that a line locked into the data cache remains locked even after it has been subjected to an invalidate-entry operation. This will leave an unusable line in the cache until a global unlock has occurred. For this reason, do not use these commands on locked lines.



This same register also provides the command to invalidate the entire data cache and mini-data cache. Refer to [Table 20, “Cache Functions” on page 104](#) for a listing of the commands. These global invalidate commands have no effect on lines locked in the data cache. Locked lines must be unlocked before they can be invalidated. This is accomplished by the Unlock Data Cache command found in [Table 22, “Cache Lock-Down Functions” on page 105](#).

A simple software routine is used to globally clean the data cache. It takes advantage of the line-allocate data cache operation, which allocates a line into the data cache. This allocation evicts any cache dirty data back to external memory. [Example 9](#) shows how data cache can be cleaned.

Example 9. Global Clean Operation

```
; Global Clean/Invalidate THE DATA CACHE
; R1 contains the virtual address of a region of cacheable memory reserved for
; this clean operation
; R0 is the loop count; Iterate 1024 times which is the number of lines in the
; data cache

;; Macro ALLOCATE performs the line-allocation cache operation on the
;; address specified in register Rx.
;;
MACRO ALLOCATE Rx
    MCR P15, 0, Rx, C7, C2, 5
ENDM

MOV R0, #1024
LOOP1:
ALLOCATE R1                ; Allocate a line at the virtual address
                           ; specified by R1.
ADD R1, R1, #32            ; Increment the address in R1 to the next cache line
SUBS R0, R0, #1            ; Decrement loop count
BNE LOOP1
;
; Clean the Mini-data Cache
; Can't use line-allocate command, so cycle 2KB of unused data through.
; R2 contains the virtual address of a region of cacheable memory reserved for
; cleaning the Mini-data Cache
; R0 is the loop count; Iterate 64 times which is the number of lines in the
; Mini-data Cache.

MOV R0, #64
LOOP2:
LDR R3, [R2], #32 ; Load and increment to next cache line
SUBS R0, R0, #1   ; Decrement loop count
BNE LOOP2
;
; Invalidate the data cache and mini-data cache
MCR P15, 0, R0, C7, C6, 0
;
```

The line-allocate operation does not require physical memory to exist at the virtual address specified by the instruction, since it does not generate a load/fill request to external memory. Also, the line-allocate operation does not set the 32 bytes of data associated with the line to any known value. Reading this data will produce unpredictable results.

The line-allocate command will not operate on the mini Data Cache, so system software must clean this cache by reading 2KByte of contiguous unused data into it. This data must be unused and reserved for this purpose so that it will not already be in the cache. It must reside in a page that is marked as mini Data Cache cacheable (see [“New Page Attributes” on page 175](#)).



The time it takes to execute a global clean operation depends on the number of dirty lines in cache.

3.4.3 Reconfiguring the Data Cache as Data RAM

Software has the ability to lock tags associated with 32-byte lines in the data cache, thus creating the appearance of data RAM. Any subsequent access to this line will always hit the cache unless it is invalidated. Once a line is locked into the data cache it is no longer available for cache allocation on a line fill. Up to 28 lines in each set can be reconfigured as data RAM, such that the maximum data RAM size is 28 Kbytes for the 32-Kbyte.

Hardware does not support locking lines into the mini-data cache; any attempt to do this will produce unpredictable results.

There are two methods for locking tags into the data cache; the method of choice depends on the application. One method is used to lock data that resides in external memory into the data cache and the other method is used to re-configure lines in the data cache as data RAM. Locking data from external memory into the data cache is useful for lookup tables, constants, and any other data that is frequently accessed. Re-configuring a portion of the data cache as data RAM is useful when an application needs scratch memory (bigger than the register file can provide) for frequently used variables. These variables may be strewn across memory, making it advantageous for software to pack them into data RAM memory.

Code examples for these two applications are shown in [Example 10 on page 93](#) and [Example 11 on page 94](#). The difference between these two routines is that [Example 10 on page 93](#) actually requests the entire line of data from external memory and [Example 11 on page 94](#) uses the line-allocate operation to lock the tag into the cache. No external memory request is made, which means software can map any unallocated area of memory as data RAM. However, the line-allocate operation does validate the target address with the MMU, so system software must ensure that the memory has a valid descriptor in the page table.

Another item to note in [Example 11 on page 94](#) is that the 32 bytes of data located in a newly allocated line in the cache must be initialized by software before it can be read. The line allocate operation does not initialize the 32 bytes and therefore reading from that line will produce unpredictable results.

In both examples, the code drains the pending loads before and after locking data. This step ensures that outstanding loads do not end up in the wrong place -- either unintentionally locked into the cache or mistakenly left out in the proverbial cold. Note also that a drain operation has been placed after the operation that locks the tag into the cache. This drains ensures predictable results if a programmer tries to lock more than 28 lines in a set; the tag will get allocated in this case but not locked into the cache.



Example 10. Locking Data into Data Cache

```

; R1 contains the virtual address of a region of memory to lock,
; configured with C=1 and B=1
; R0 is the number of 32-byte lines to lock into the data cache. In this
; example 16 lines of data are locked into the cache.
; MMU and data cache are enabled prior to this code.

MACRO DRAIN
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores
ENDM

DRAIN

MOV R2, #0x1
MCR P15,0,R2,C9,C2,0             ; Put the data cache in lock mode
CPWAIT
MOV R0, #16
LOOP1:
MCR P15,0,R1,C7,C10,1           ; Write back the line if it is dirty in the cache
MCR P15,0,R1, C7,C6,1           ; Flush/Invalidate the line from the cache
LDR R2, [R1], #32               ; Load and lock 32 bytes of data located at [R1]
                                ; into the data cache. Post-increment the address
                                ; in R1 to the next cache line.
SUBS R0, R0, #1                 ; Decrement loop count
BNE LOOP1

                                ; Turn off data cache locking
MOV R2, #0x0
MCR P15,0,R2,C9,C2,0             ; Take the data cache out of lock mode.
CPWAIT

```




Example 11. Creating Data RAM

```
; R1 contains the virtual address of a region of memory to configure as data RAM,  
; which is aligned on a 32-byte boundary.  
; MMU is configured so that the memory region is cacheable.  
; R0 is the number of 32-byte lines to designate as data RAM. In this example 16  
; lines of the data cache are re-configured as data RAM.  
; The inner loop is used to initialize the newly allocated lines  
; MMU and data cache are enabled prior to this code.  
  
MACRO ALLOCATE Rx  
    MCR P15, 0, Rx, C7, C2, 5  
ENDM  
  
MACRO DRAIN  
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores  
ENDM  
  
DRAIN  
MOV R4, #0x0  
MOV R5, #0x0  
MOV R2, #0x1  
MCR P15, 0, R2, C9, C2, 0        ; Put the data cache in lock mode  
CPWAIT  
  
MOV R0, #16  
LOOP1:  
ALLOCATE R1                      ; Allocate and lock a tag into the data cache at  
                                ; address [R1].  
; initialize 32 bytes of newly allocated line  
DRAIN  
STRD R4, [R1], #8                ;  
STRD R4, [R1], #8                ;  
STRD R4, [R1], #8                ;  
STRD R4, [R1], #8                ;  
  
SUBS R0, R0, #1                  ; Decrement loop count  
BNE LOOP1  
; Turn off data cache locking  
  
DRAIN                            ; Finish all pending operations  
  
MOV R2, #0x0  
MCR P15, 0, R2, C9, C2, 0; Take the data cache out of lock mode.  
CPWAIT
```

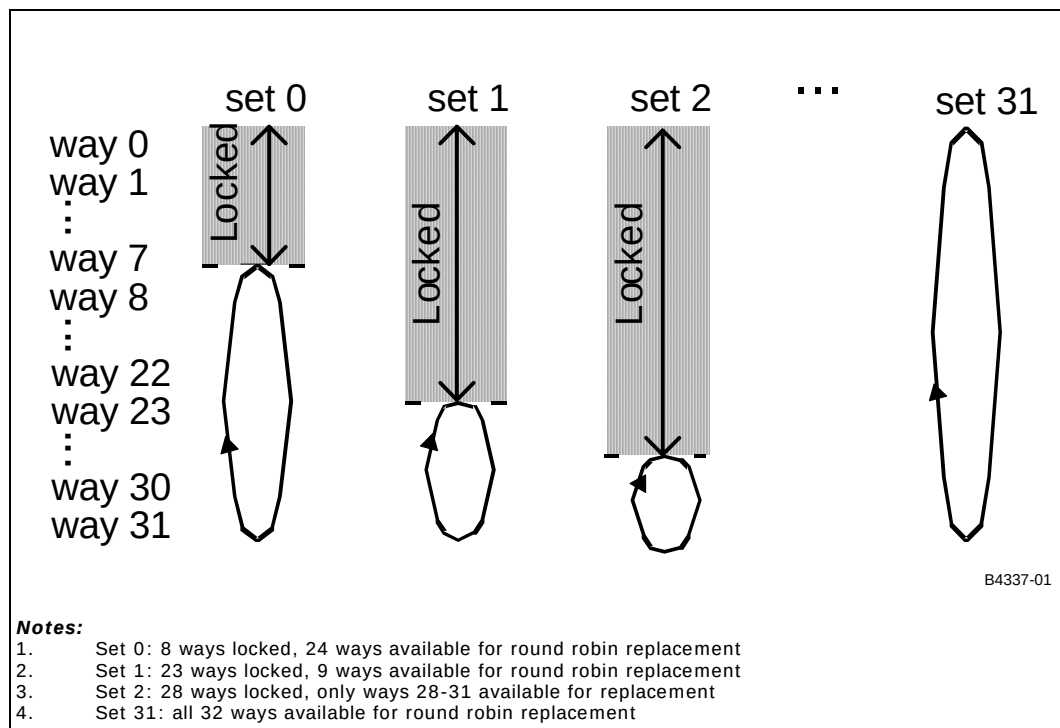
Tags can be locked into the data cache by enabling the data cache lock mode bit located in coprocessor 15, register 9. (See [Table 22, “Cache Lock-Down Functions” on page 105](#) for the exact command.) Once enabled, any new lines allocated into the data cache will be locked down.

Note that the **PLD** instruction will not affect the cache contents if it encounters an error while executing. For this reason, system software should ensure the memory address used in the **PLD** is correct. If this cannot be ascertained, replace the **PLD** with a **LDR** instruction that targets a scratch register.

Lines are locked into a set starting at way0 and may progress up to way 27; which set a line gets locked into depends on the set index of the virtual address of the request. [Figure 7, “Locked Line Effect on Round-Robin Replacement” on page 82](#) is an example of where lines of code may be locked into the cache along with how the round-robin pointer is affected.



Figure 12. Locked Line Effect on Round-Robin Replacement



Software can lock down data located at different memory locations. This may cause some sets to have more locked lines than others as shown in [Figure 7](#).

Lines are unlocked in the data cache by performing an unlock operation. See [“Register 9: Cache Lock Down” on page 105](#) for more information about locking and unlocking the data cache.

Before locking, the programmer must ensure that no part of the target data range is already resident in the cache. The Intel XScale processor will not prefetch such data, which will result in it not being locked into the cache. If there is any doubt as to the location of the targeted memory data, the cache should be cleaned and invalidated to prevent this scenario. If the cache contains a locked region which the programmer wishes to lock again, then the cache must be unlocked before being cleaned and invalidated.

See [“Terminology and Conventions” on page 39](#) for a definition of coalescing.

The write buffer is always enabled which means stores to external memory will be buffered. The K bit in the Auxiliary Control Register (CP15, register 1) is a global enable/disable for allowing coalescing in the write buffer. When this bit disables coalescing, no coalescing will occur regardless the value of the page attributes. If this bit enables coalescing, the page attributes X, C, and B are examined to see if coalescing is enabled for each region of memory.

All reads and writes to external memory occur in program order when coalescing is disabled in the write buffer. If coalescing is enabled in the write buffer, writes may occur out of program order to external memory. Program correctness is maintained in this case by comparing all store requests with all the valid entries in the fill buffer.



The write buffer and fill buffer support a drain operation, such that before the next instruction executes, all Intel XScale processor data requests to external memory have completed. See [Table 20, “Cache Functions” on page 104](#) for the exact command.

Writes to a region marked non-cacheable/non-bufferable (page attributes C, B, and X all 0) will cause execution to stall until the write completes.

If software is running in a privileged mode, it can explicitly drain all buffered writes. For details on this operation, see the description of Drain Write Buffer in [“Register 7: Cache Functions” on page 103](#).

3.5 Configuration

This section describes the System Control Coprocessor (CP15) and coprocessor 14 (CP14). CP15 configures the MMU, caches, buffers and other system attributes. Where possible, the definition of CP15 follows the definition of the Intel® StrongARM* products. CP14 contains the performance monitor registers, clock and power management registers and the debug registers.

CP15 is accessed through **MRC** and **MCR** coprocessor instructions and allowed only in privileged mode. Any access to CP15 in user mode or with **LDC** or **STC** coprocessor instructions will cause an undefined instruction exception.

All CP14 registers can be accessed through **MRC** and **MCR** coprocessor instructions. **LDC** and **STC** coprocessor instructions can only access the clock and power management registers, and the debug registers. The performance monitoring registers can't be accessed by **LDC** and **STC** because CRm != 0x0. Access to all registers is allowed only in privileged mode. Any access to CP14 in user mode will cause an undefined instruction exception.

Coprocessors, CP15 and CP14, on the Intel XScale processor do not support access via **CDP**, **MRRC**, or **MCRR** instructions. An attempt to access these coprocessors with these instructions will result in an Undefined Instruction exception.

Many of the MCR commands available in CP15 modify hardware state sometime after execution. A software sequence is available for those wishing to determine when this update occurs and can be found in [“Additions to CP15 Functionality” on page 176](#).

Like certain other Intel® StrongARM* architecture products, the Intel XScale processor includes an extra level of virtual address translation in the form of a PID (Process ID) register and associated logic. For a detailed description of this facility, see [“Register 13: Process ID” on page 106](#). Privileged code needs to be aware of this facility because, when interacting with CP15, some addresses are modified by the PID and others are not.

An address that has yet to be modified by the PID (“PIDified”) is known as a *virtual address* (VA). An address that has been through the PID logic, but not translated into a physical address, is a *modified virtual address* (MVA). Non-privileged code always deals with VAs, while privileged code that programs CP15 occasionally needs to use MVAs.

The format of **MRC** and **MCR** is shown in [Table 9](#).

cp_num is defined for CP15, CP14 and CP0 on the Intel XScale processor. CP0 supports instructions specific for DSP and is described in [“Programming Model” on page 167](#)

Unless otherwise noted, unused bits in coprocessor registers have unpredictable values when read. For compatibility with future implementations, software should not rely on the values in those bits.



Table 9. MRC/MCR Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
cond				1	1	1	0	opcode_1				n	CRn				Rd				cp_num				opcode_2				1	CRm			

Bits	Description	Notes
31:28	cond - Intel® StrongARM® condition codes	-
23:21	opcode_1 - Reserved	Should be programmed to zero for future compatibility
20	n - Read or write coprocessor register 0 = MCR 1 = MRC	-
19:16	CRn - specifies which coprocessor register	-
15:12	Rd - General-Purpose Register, R0..R15	-
11:8	cp_num - coprocessor number	Intel XScale® Processor defines three coprocessors: 0b1111 = CP15 0b1110 = CP14 0x0000 = CP0 Note: Mappings are implementation defined for all coprocessors below CP14 and above CP0. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:5	opcode_2 - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.
3:0	CRm - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.

The format of **LDC** and **STC** for CP14 is shown in Table 10. **LDC** and **STC** follow the programming notes in the *ARM® Architecture Reference Manual*. Note that access to CP15 with **LDC** and **STC** will cause an undefined exception.

LDC and **STC** transfer a single 32-bit word between a coprocessor register and memory. These instructions do not allow the programmer to specify values for **opcode_1**, **opcode_2**, or **Rm**; those fields implicitly contain zero, which means the performance monitoring registers are not accessible.

Table 10. LDC/STC Format when Accessing CP14 (Sheet 1 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8_bit_word_offset									

Bits	Description	Notes
31:28	cond - Intel® StrongARM® condition codes	-
24:23,21	P, U, W - specifies 1 of 3 addressing modes identified by addressing mode 5 in the <i>ARM® Architecture Reference Manual</i> .	-
22	N - should be 0 for CP14 coprocessors. Setting this bit to 1 has will have an undefined effect.	
20	L - Load or Store 0 = STC 1 = LDC	-

**Table 10. LDC/STC Format when Accessing CP14 (Sheet 2 of 2)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8_bit_word_offset							

Bits	Description	Notes
19:16	Rn - specifies the base register	-
15:12	CRd - specifies the coprocessor register	-
11:8	cp_num - coprocessor number	Intel XScale® Processor defines the following: 0b1111 = Undefined Exception 0b1110 = CP14 Note: Mappings are implementation defined for all coprocessors below CP13. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:0	8-bit word offset	-

3.5.1 CP15 Registers

Table 11 lists the CP15 registers implemented in the IXP45X/IXP46X network processors.

Table 11. CP15 Registers

Register (CRn)	Opcode_2	Access	Description
0	0	Read / Write-Ignored	ID
0	1	Read / Write-Ignored	Cache Type
1	0	Read / Write	Control
1	1	Read / Write	Auxiliary Control
2	0	Read / Write	Translation Table Base
3	0	Read / Write	Domain Access Control
4	-	Unpredictable	(Reserved)
5	0	Read / Write	Fault Status
6	0	Read / Write	Fault Address
7	0	Read-unpredictable / Write	Cache Operations
8	0	Read-unpredictable / Write	TLB Operations
9	0	Read / Write	Cache Lock Down
10	0	Read-unpredictable / Write	TLB Lock Down
11 - 12	-	Unpredictable	(Reserved)
13	0	Read / Write	Process ID (PID)
14	0	Read / Write	Breakpoint Registers
15	0	Read / Write	(CRm = 1) CP Access

3.5.1.1 Register 0: ID & Cache Type Registers

Register 0 houses two read-only register that are used for part identification: an ID register and a cache type register.



The ID Register is selected when *opcode_2*=0. This register returns the code for the IXP45X/IXP46X network processors.

Table 12. ID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	0	0	0	1	0	1																	
																Core Gen		Core Revision		Product Number				Product Revision							

reset value: As Shown

Bits	Access	Description
31:24	Read / Write Ignored	Implementation trademark (0x69 = 'i' = Intel Corporation)
23:16	Read / Write Ignored	Architecture version = Intel® StrongARM® Version 5TE (0x05 will be value returned)
15:13	Read / Write Ignored	Core Generation 0b010 = Intel XScale® Processor This field reflects a specific set of architecture features supported by the core. If new features are added/deleted/modified this field will change.
12:10	Read / Write Ignored	Core Revision: This field reflects revisions of core generations. Differences may include errata that dictate different operating conditions, software work-around, etc. Value returned will be 000b
9:4	Read / Write Ignored	Product Number for: IXP45X/IXP46X network processors 100000b
3:0	Read / Write Ignored	Product Revision for: IXP45X/IXP46X network processors 0000b

The Cache Type Register is selected when *opcode_2*=1 and describes the cache configuration of the Intel XScale processor.

Table 13. Cache Type Register (Sheet 1 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	0																					
								Dsize								Isize															

reset value: As Shown

Bits	Access	Description
31:29	Read-as-Zero / Write Ignored	Reserved
28:25	Read / Write Ignored	Cache class = 0b0101 The caches support locking, write back and round-robin replacement. They do not support address by index.
24	Read / Write Ignored	Harvard Cache
23:21	Read-as-Zero / Write Ignored	Reserved
20:18	Read / Write Ignored	Data Cache Size (Dsize) 0b110 = 32 KB
17:15	Read / Write Ignored	Data cache associativity = 0b101 = 32-way
14	Read-as-Zero / Write Ignored	Reserved
13:12	Read / Write Ignored	Data cache line length = 0b10 = 8 words/line
11:9	Read-as-Zero / Write Ignored	Reserved
8:6	Read / Write Ignored	Instruction cache size (Isize) 0b110 = 32 KB

**Table 13. Cache Type Register (Sheet 2 of 2)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	0	Dsize	1	0	1	0	1	0	0	0	0	Isize	1	0	1	0	1	0				

reset value: As Shown

Bits	Access	Description
5:3	Read / Write Ignored	Instruction cache associativity = 0b101 = 32-way
2	Read-as-Zero / Write Ignored	Reserved
1:0	Read / Write Ignored	Instruction cache line length = 0b10 = 8 words/line

3.5.1.2 Register 1: Control and Auxiliary Control Registers

Register 1 is made up of two registers, one that is compliant with Intel® StrongARM® Version 5TE and referred by *opcode_2* = 0x0, and the other which is specific to the Intel XScale processor is referred by *opcode_2* = 0x1. The latter is known as the Auxiliary Control Register.

The Exception Vector Relocation bit (bit 13 of the Intel® StrongARM® control register) allows the vectors to be mapped into high memory rather than their default location at address 0. This bit is readable and writable by software. If the MMU is enabled, the exception vectors will be accessed via the usual translation method involving the PID register (see “[Register 13: Process ID](#)” on page 106) and the TLBs. To avoid automatic application of the PID to exception vector accesses, software may relocate the exceptions to high memory.

Table 14. Intel® StrongARM® Control Register (Sheet 1 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																		V	I	Z	0	R	S	B	1	1	1	1	C	A	M

reset value: writable bits set to 0

Bits	Access	Description
31:14	Read-Unpredictable / Write-as-Zero	Reserved
13	Read / Write	Exception Vector Relocation (V). 0 = Base address of exception vectors is 0x0000,0000 1 = Base address of exception vectors is 0xFFFF,0000
12	Read / Write	Instruction Cache Enable/Disable (I) 0 = Disabled 1 = Enabled
11	Read / Write	Branch Target Buffer Enable (Z) 0 = Disabled 1 = Enabled
10	Read-as-Zero / Write-as-Zero	Reserved
9	Read / Write	ROM Protection (R) This selects the access checks performed by the memory management unit. See the <i>ARM® Architecture Reference Manual</i> for more information.
8	Read / Write	System Protection (S) This selects the access checks performed by the memory management unit. See the <i>ARM® Architecture Reference Manual</i> for more information.
7	Read / Write	Big- / Little-Endian (B) 0 = Little-endian operation 1 = Big-endian operation
6:3	Read-as-One / Write-as-One	= 0b1111

**Table 14. Intel® StrongARM® Control Register (Sheet 2 of 2)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																		V	I	Z	0	R	S	B	1	1	1	1	C	A	M

reset value: writable bits set to 0

Bits	Access	Description
2	Read / Write	Data cache enable/disable (C) 0 = Disabled 1 = Enabled
1	Read / Write	Alignment fault enable/disable (A) 0 = Disabled 1 = Enabled
0	Read / Write	Memory management unit enable/disable (M) 0 = Disabled 1 = Enabled

The mini-data cache attribute bits, in the Auxiliary Control Register, are used to control the allocation policy for the mini-data cache and whether it will use write-back caching or write-through caching.

The configuration of the mini-data cache should be setup before any data access is made that may be cached in the mini-data cache. Once data is cached, software must ensure that the mini-data cache has been cleaned and invalidated before the mini-data cache attributes can be changed.

Table 15. Auxiliary Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																						PTEX		MD	C	B	P	K			

reset value: writable bits set to 0

Bits	Access	Description
31:6	Read-Unpredictable / Write-as-Zero	Reserved
5:4	Read / Write	Mini Data Cache Attributes (MD) All configurations of the Mini-data cache are cacheable, stores are buffered in the write buffer and stores will be coalesced in the write buffer as long as coalescing is globally enable (bit 0 of this register). 0b00 = Write back, Read allocate 0b01 = Write back, Read/Write allocate 0b10 = Write through, Read allocate 0b11 = Unpredictable
3:2	Read-Unpredictable/ Write-as-Zero	(Reserved)
1	Read/Write	Page Table Memory Attribute (P) This is a request to the core memory bus for a hardware page table walk. An ASSP may use this bit to direct the external bus controller to perform some special operation on the memory access.
0	Read / Write	Write Buffer Coalescing Disable (K) This bit globally disables the coalescing of all stores in the write buffer no matter what the value of the Cacheable and Bufferable bits are in the page table descriptors. 0 = Enabled 1 = Disabled



3.5.1.3 Register 2: Translation Table Base Register

Table 16. Translation Table Base Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Translation Table Base																															
reset value: unpredictable																															
Bits		Access										Description																			
31:14		Read / Write										Translation Table Base - Physical address of the base of the first-level table																			
13:0		Read-unpredictable / Write-as-Zero										Reserved																			

3.5.1.4 Register 3: Domain Access Control Register

Table 17. Domain Access Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0																
reset value: unpredictable																															
Bits		Access														Description															
31:0		Read / Write														Access permissions for all 16 domains - The meaning of each field can be found in the ARM* Architecture Reference Manual.															

3.5.1.5 Register 4: Reserved

Register 4 is reserved. Reading and writing this register yields unpredictable results.

3.5.1.6 Register 5: Fault Status Register

The Fault Status Register (FSR) indicates which fault has occurred, which could be either a prefetch abort or a data abort. Bit 10 extends the encoding of the status field for prefetch aborts and data aborts. The definition of the extended status field is found in “[Event Architecture](#)” on page 177. Bit 9 indicates that a debug event occurred and the exact source of the event is found in the debug control and status register (CP14, register 10). When bit 9 is set, the domain and extended status field are undefined.

Upon entry into the prefetch abort or data abort handler, hardware will update this register with the source of the exception. Software is not required to clear these fields.

**Table 18. Fault Status Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																						X	D	0	Domain				Status		
reset value: unpredictable																															
Bits		Access										Description																			
31:11		Read-unpredictable / Write-as-Zero										Reserved																			
10		Read / Write										Status Field Extension (X) This bit is used to extend the encoding of the Status field, when there is a prefetch abort and when there is a data abort. The definition of this field can be found in " Event Architecture " on page 177																			
9		Read / Write										Debug Event (D) This flag indicates a debug event has occurred and that the cause of the debug event is found in the MOE field of the debug control register (CP14, register 10)																			
8		Read-as-zero / Write-as-Zero										= 0																			
7:4		Read / Write										Domain - Specifies which of the 16 domains was being accessed when a data abort occurred																			
3:0		Read / Write										Status - Type of data access being attempted																			

3.5.1.7 Register 6: Fault Address Register

Table 19. Fault Address Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fault Virtual Address																															
reset value: unpredictable																															
Bits	Access										Description																				
31:0	Read / Write										Fault Virtual Address - Contains the MVA of the data access that caused the memory abort																				

3.5.1.8 Register 7: Cache Functions

All the functions defined in existing Intel® StrongARM® products appear here. The Intel XScale processor adds other functions as well. This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

The Drain Write Buffer function not only drains the write buffer but also drains the fill buffer.

The Intel XScale processor does not check permissions on addresses supplied for cache or TLB functions. Due to the fact only privileged software may execute these functions, full accessibility is assumed. Cache functions will not generate any of the following:

- Translation faults
- Domain faults
- Permission faults

The invalidate instruction cache line command does not invalidate the BTB. If software invalidates a line from the instruction cache and modifies the same location in external memory, it needs to invalidate the BTB also. Not invalidating the BTB in this case may cause unpredictable results.



Disabling/enabling a cache has no effect on contents of the cache: valid data stays valid, locked items remain locked. All operations defined in [Table 20](#) work regardless of whether the cache is enabled or disabled.

Since the Clean DCache Line function reads from the data cache, it is capable of generating a parity fault. The other operations will not generate parity faults.

Table 20. Cache Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D cache & BTB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c7, c7, 0
Invalidate I cache & BTB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 0
Invalidate I cache line	0b001	0b0101	MVA	MCR p15, 0, Rd, c7, c5, 1
Invalidate D cache	0b000	0b0110	Ignored	MCR p15, 0, Rd, c7, c6, 0
Invalidate D cache line	0b001	0b0110	MVA	MCR p15, 0, Rd, c7, c6, 1
Clean D cache line	0b001	0b1010	MVA	MCR p15, 0, Rd, c7, c10, 1
Drain Write (& Fill) Buffer	0b100	0b1010	Ignored	MCR p15, 0, Rd, c7, c10, 4
Invalidate Branch Target Buffer	0b110	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 6
Allocate Line in the Data Cache	0b101	0b0010	MVA	MCR p15, 0, Rd, c7, c2, 5

The line-allocate command allocates a tag into the data cache specified by bits [31:5] of Rd. If a valid dirty line (with a different MVA) already exists at this location it will be evicted. The 32 bytes of data associated with the newly allocated line are not initialized and therefore will generate unpredictable results if read.

Line allocate command may be used for cleaning the entire data cache on a context switch and also when reconfiguring portions of the data cache as data RAM. In both cases, Rd is a virtual address that maps to some non-existent physical memory. When creating data RAM, software must initialize the data RAM before read accesses can occur. Specific uses of these commands can be found in [Chapter 3.0, “Data Cache”](#).

Other items to note about the line-allocate command are:

- It forces all pending memory operations to complete.
- Bits [31:5] of Rd is used to specific the virtual address of the line to be allocated into the data cache.
- If the targeted cache line is already resident, this command has no effect.
- The command cannot be used to allocate a line in the mini Data Cache.
- The newly allocated line is not marked as “dirty” so it will never get evicted. However, if a valid store is made to that line it will be marked as “dirty” and will get written back to external memory if another line is allocated to the same cache location. This eviction will produce unpredictable results.

To avoid this situation, the line-allocate operation should only be used if one of the following can be guaranteed:

- The virtual address associated with this command is not one that will be generated during normal program execution. This is the case when line-allocate is used to clean/invalidate the entire cache.
- The line-allocate operation is used only on a cache region destined to be locked. When the region is unlocked, it must be invalidated before making another data access.

3.5.1.9 Register 8: TLB Operations

Disabling/enabling the MMU has no effect on the contents of either TLB: valid entries stay valid, locked items remain locked. All operations defined in [Table 21](#) work regardless of whether the TLB is enabled or disabled.



This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 21. TLB Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D TLB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c8, c7, 0
Invalidate I TLB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c8, c5, 0
Invalidate I TLB entry	0b001	0b0101	MVA	MCR p15, 0, Rd, c8, c5, 1
Invalidate D TLB	0b000	0b0110	Ignored	MCR p15, 0, Rd, c8, c6, 0
Invalidate D TLB entry	0b001	0b0110	MVA	MCR p15, 0, Rd, c8, c6, 1

3.5.1.10 Register 9: Cache Lock Down

Register 9 is used for locking down entries into the instruction cache and data cache. (The protocol for locking down entries can be found in [Chapter 3.0, “Data Cache”](#).)

[Table 22](#) shows the command for locking down entries in the instruction and data cache. The entry to lock in the instruction cache is specified by the virtual address in Rd. The data cache locking mechanism follows a different procedure than the instruction cache. The data cache is placed in lock down mode such that all subsequent fills to the data cache result in that line being locked in, as controlled by [Table 23](#).

Lock/unlock operations on a disabled cache have an undefined effect.

Read and write access is allowed to the data cache lock register bit[0]. All other accesses to register 9 should be write-only; reads, as with an MRC, have an undefined effect.

Table 22. Cache Lock-Down Functions

Function	opcode_2	CRm	Data	Instruction
Fetch and Lock I cache line	0b000	0b0001	MVA	MCR p15, 0, Rd, c9, c1, 0
Unlock Instruction cache	0b001	0b0001	Ignored	MCR p15, 0, Rd, c9, c1, 1
Read data cache lock register	0b000	0b0010	Read lock mode value	MCR p15, 0, Rd, c9, c2, 0
Write data cache lock register	0b000	0b0010	Set/Clear lock mode	MCR p15, 0, Rd, c9, c2, 0
Unlock Data Cache	0b001	0b0010	Ignored	MCR p15, 0, Rd, c9, c2, 1

Table 23. Data Cache Lock Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															L

reset value: writable bits set to 0

Bits	Access	Description
31:1	Read-unpredictable / Write-as-Zero	Reserved
0	Read / Write	Data Cache Lock Mode (L) 0 = No locking occurs 1 = Any fill into the data cache while this bit is set gets locked in



3.5.1.11 Register 10: TLB Lock Down

Register 10 is used for locking down entries into the instruction TLB, and data TLB. (The protocol for locking down entries can be found in [Chapter 3.0, “Memory Management Unit”](#).) Lock/unlock operations on a TLB when the MMU is disabled have an undefined effect.

This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

[Table 24](#) shows the command for locking down entries in the instruction TLB, and data TLB. The entry to lock is specified by the virtual address in Rd.

Table 24. TLB Lockdown Functions

Function	opcode_2	CRm	Data	Instruction
Translate and Lock I TLB entry	0b000	0b0100	MVA	MCR p15, 0, Rd, c10, c4, 0
Translate and Lock D TLB entry	0b000	0b1000	MVA	MCR p15, 0, Rd, c10, c8, 0
Unlock I TLB	0b001	0b0100	Ignored	MCR p15, 0, Rd, c10, c4, 1
Unlock D TLB	0b001	0b1000	Ignored	MCR p15, 0, Rd, c10, c8, 1

3.5.1.12 Register 11-12: Reserved

These registers are reserved. Reading and writing them yields unpredictable results.

3.5.1.13 Register 13: Process ID

The Intel XScale processor supports the remapping of virtual addresses through a Process ID (PID) register. This remapping occurs before the instruction cache, instruction TLB, data cache and data TLB are accessed. The PID register controls when virtual addresses are remapped and to what value.

The PID register is a 7-bit value that is ORed with bits 31:25 of the virtual address when they are zero. This action effectively remaps the address to one of 128 “slots” in the 4 Gbytes of address space. If bits 31:25 are not zero, no remapping occurs. This feature is useful for operating system management of processes that may map to the same virtual address space. In those cases, the virtually mapped caches on the Intel XScale processor would not require invalidating on a process switch.

Table 25. Accessing Process ID

Function	opcode_2	CRm	Instruction
Read Process ID Register	0b000	0b0000	MRC p15, 0, Rd, c13, c0, 0
Write Process ID Register	0b000	0b0000	MCR p15, 0, Rd, c13, c0, 0

Table 26. Process ID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Process ID																															
reset value: 0x0000,0000																															
Bits		Access										Description																			
31:25		Read / Write										Process ID - This field is used for remapping the virtual address when bits 31-25 of the virtual address are zero.																			
24:0		Read-as-Zero / Write-as-Zero										Reserved - Should be programmed to zero for future compatibility																			



3.5.1.14 The PID Register Affect On Addresses

All addresses generated and used by User Mode code are eligible for being “PIDified” as described in the previous section. Privileged code, however, must be aware of certain special cases in which address generation does not follow the usual flow.

The PID register is not used to remap the virtual address when accessing the Branch Target Buffer (BTB). Any writes to the PID register invalidate the BTB, which prevents any virtual addresses from being double mapped between two processes.

A breakpoint address (see “[Register 14: Breakpoint Registers](#)” on page 107) must be expressed as an MVA when written to the breakpoint register. This requirement means the value of the PID must be combined appropriately with the address before it is written to the breakpoint register. All virtual addresses in translation descriptors (see “[Memory Management Unit](#)” on page 69) are MVAs.

3.5.1.15 Register 14: Breakpoint Registers

The Intel XScale processor contains two instruction breakpoint address registers (IBCR0 and IBCR1), one data breakpoint address register (DBR0), one configurable data mask/address register (DBR1), and one data breakpoint control register (DBCON). The Intel XScale processor also supports a 256-entry, trace buffer that records program execution information. The registers to control the trace buffer are located in CP14.

Refer to “[Software Debug](#)” on page 111 for more information on these features of the Intel XScale processor.

Table 27. Accessing the Debug Registers

Function	opcode_2	CRm	Instruction
Access Instruction Breakpoint Control Register 0 (IBCR0)	0b000	0b1000	MRC p15, 0, Rd, c14, c8, 0 ; read MCR p15, 0, Rd, c14, c8, 0 ; write
Access Instruction Breakpoint Control Register 1 (IBCR1)	0b000	0b1001	MRC p15, 0, Rd, c14, c9, 0 ; read MCR p15, 0, Rd, c14, c9, 0 ; write
Access Data Breakpoint Address Register (DBR0)	0b000	0b0000	MRC p15, 0, Rd, c14, c0, 0 ; read MCR p15, 0, Rd, c14, c0, 0 ; write
Access Data Mask/Address Register (DBR1)	0b000	0b0011	MRC p15, 0, Rd, c14, c3, 0 ; read MCR p15, 0, Rd, c14, c3, 0 ; write
Access Data Breakpoint Control Register (DBCON)	0b000	0b0100	MRC p15, 0, Rd, c14, c4, 0 ; read MCR p15, 0, Rd, c14, c4, 0 ; write

3.5.1.16 Register 15: Coprocessor Access Register

This register is selected when *opcode_2* = 0 and *CRm* = 1.

This register controls access rights to all the coprocessors in the system except for CP15 and CP14. Both CP15 and CP14 can only be accessed in privilege mode. This register is accessed with an MCR or MRC with the CRm field set to 1.

This register controls access to CP0, atypical use for this register is for an operating system to control resource sharing among applications. Initially, all applications are denied access to shared resources by clearing the appropriate coprocessor bit in the Coprocessor Access Register. An application may request the use of a shared resource (e.g., the accumulator in CP0) by issuing an access to the resource, which will result in an undefined exception. The operating system may grant access to this coprocessor by setting the appropriate bit in the Coprocessor Access Register and return to the application where the access is retried.



Sharing resources among different applications requires a state saving mechanism. Two possibilities are:

- The operating system, during a context switch, could save the state of the coprocessor if the last executing process had access rights to the coprocessor.
- The operating system, during a request for access, saves off the old coprocessor state and saves it with last process to have access to it.

Under both scenarios, the OS needs to restore state when a request for access is made. This means the OS has to maintain a list of what processes are modifying CP0 and their associated state.

Example 12. Disallowing access to CP0

```
;; The following code clears bit 0 of the CPAR.
;; This will cause the processor to fault if software
;; attempts to access CP0.

LDR R0, =0x3FFE                ; bit 0 is clear
MCR P15, 0, R0, C15, C1, 0    ; move to CPAR
CPWAIT                        ; wait for effect
```

Table 28. Coprocessor Access Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																0	0	CP13	CP12	CP11	CP10	CP9	CP8	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0

reset value: 0x0000,0000

Bits	Access	Description
31:16	Read-unpredictable / Write-as-Zero	Reserved - Should be programmed to zero for future compatibility
15:14	Read-as-Zero/Write-as-Zero	Reserved - Should be programmed to zero for future compatibility
13:1	Read / Write	Coprocessor Access Rights - Each bit in this field corresponds to the access rights for each coprocessor.
0	Read / Write	Coprocessor Access Rights - This bit corresponds to the access rights for CP0. 0 = Access denied. Any attempt to access the corresponding coprocessor will generate an undefined exception. 1 = Access allowed. Includes read and write accesses.

3.5.2 CP14 Registers

Table 29 lists the CP14 registers implemented in the Intel XScale processor.

**Table 29. CP14 Registers**

Description	Access	Register# (CRn)	Register# (CRm)
Performance Monitoring	Read / Write	0,1,4,5,8	1
		0-3	2
Clock and Power Management	Read / Write	6-7	0
Software Debug	Read / Write	8-15	0

All other registers are reserved in CP14. Reading and writing them yields unpredictable results.

3.5.2.1 Performance Monitoring Registers

The performance monitoring unit contains a control register (PMNC), a clock counter (CCNT), interrupt enable register (INTEN), overflow flag register (FLAG), event selection register (EVTSEL) and four event counters (PMN0 through PMN3). The format of these registers can be found in [“Performance Monitoring” on page 157](#), along with a description on how to use the performance monitoring facility.

Opcode_2 should be zero on all accesses.

These registers can't be accessed by **LDC** and **STC** coprocessor instructions.

Table 30. Accessing the Performance Monitoring Registers

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

3.5.2.2 Clock and Power Management Registers

These registers contain functions for managing the core clock and power.

For the IXP45X/IXP46X network processors, these registers are not implemented and reserved for future use.

**Table 31. PWRMODE Register**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															M

reset value: writable bits set to 0

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero	Reserved
1:0	Read / Write	Mode (M) 0 = ACTIVE Never change from 00b

The Intel XScale processor clock frequency cannot be changed by software on the IXP45X/IXP46X network processors.

Table 32. Clock and Power Management

Function	Data	Instruction
Read CCLKCFG	ignored	MRC p14, 0, Rd, c6, c0, 0
Write CCLKCFG	CCLKCFG value	MCR p14, 0, Rd, c6, c0, 0

Table 33. CCLKCFG Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															CCLKCFG

reset value: unpredictable

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero always	Reserved (write as zero)

3.5.2.3 Software Debug Registers

Software debug is supported by address breakpoint registers (Coprocessor 15, register 14), serial communication over the JTAG interface and a trace buffer. Registers 8 and 9 are used for the serial interface and registers 10 through 13 support a 256 entry trace buffer. Register 14 and 15 are the debug link register and debug SPSR (saved program status register). These registers are explained in more detail in “Software Debug” on page 111.

Opcode_2 and CRm should be zero.

Table 34. Accessing the Debug Registers (Sheet 1 of 2)

Function	CRn (Register #)	Instruction
Access Transmit Debug Register (TX)	0b1000	MRC p14, 0, Rd, c8, c0, 0 MCR p14, 0, Rd, c8, c0, 0
Access Receive Debug Register (RX)	0b1001	MCR p14, 0, Rd, c9, c0, 0 MRC p14, 0, Rd, c9, c0, 0
Access Debug Control and Status Register (DBGCSR)	0b1010	MCR p14, 0, Rd, c10, c0, 0 MRC p14, 0, Rd, c10, c0, 0
Access Trace Buffer Register (TBREG)	0b1011	MCR p14, 0, Rd, c11, c0, 0 MRC p14, 0, Rd, c11, c0, 0

**Table 34. Accessing the Debug Registers (Sheet 2 of 2)**

Function	CRn (Register #)	Instruction
Access Checkpoint 0 Register (CHKPT0)	0b1100	MCR p14, 0, Rd, c12, c0, 0 MRC p14, 0, Rd, c12, c0, 0
Access Checkpoint 1 Register (CHKPT1)	0b1101	MCR p14, 0, Rd, c13, c0, 0 MRC p14, 0, Rd, c13, c0, 0
Access Transmit and Receive Debug Control Register	0b1110	MCR p14, 0, Rd, c14, c0, 0 MRC p14, 0, Rd, c14, c0, 0

3.6 Software Debug

This section describes the software debug and related features implemented in the IXP45X/IXP46X network processors, namely:

- Debug modes, registers and exceptions
- A serial debug communication link via the JTAG interface
- A trace buffer
- A mechanism to load the instruction cache through JTAG
- Debug Handler SW requirements and suggestions

3.6.1 Definitions

Debug handler: Debug handler is event handler that runs on the IXP45X/IXP46X network processors, when a debug event occurs.

Debugger: The debugger is software that runs on a host system outside of the IXP45X/IXP46X network processors.

3.6.2 Debug Registers

CP15 Registers

CRn = 14; CRm = 8: instruction breakpoint register 0 (IBCR0)

CRn = 14; CRm = 9: instruction breakpoint register 1 (IBCR1)

CRn = 14; CRm = 0: data breakpoint register 0 (DBR0)

CRn = 14; CRm = 3: data breakpoint register 1 (DBR1)

CRn = 14; CRm = 4: data breakpoint control register (DBCON)

CP15 registers are accessible using MRC and MCR. CRn and CRm specify the register to access. The opcode_1 and opcode_2 fields are not used and should be set to 0.

CP14 Registers

CRn = 8; CRm = 0: TX Register (TX)

CRn = 9; CRm = 0: RX Register (RX)

CRn = 10; CRm = 0: Debug Control and Status Register (DCSR)

CRn = 11; CRm = 0: Trace Buffer Register (TBREG)

CRn = 12; CRm = 0: Checkpoint Register 0 (CHKPT0)

CRn = 13; CRm = 0: Checkpoint Register 1 (CHKPT1)

CRn = 14; CRm = 0: TXRX Control Register (TXRXCTRL)

CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers will cause an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.



Software access to all debug registers must be done in privileged mode. User mode access will generate an undefined instruction exception. Specifying registers which do not exist has unpredictable results.

The TX and RX registers, certain bits in the TXRXCTRL register, and certain bits in the DCSR can be accessed by a debugger through the JTAG interface.

3.6.3 Debug Modes

The debug unit for the IXP45X/IXP46X network processors, when used with a debugger application, allows software running on the IXP45X/IXP46X network processors' target to be debugged. The debug unit allows the debugger to stop program execution and re-direct execution to a debug handling routine. Once program execution has stopped, the debugger can examine or modify processor state, coprocessor state, or memory. The debugger can then restart execution of the application.

On the IXP45X/IXP46X network processors, one of two debug modes can be entered:

- Halt mode
- Monitor mode

3.6.3.1 Halt Mode

When the debug unit is configured for halt mode, the reset vector is overloaded to serve as the debug vector. A new processor mode, DEBUG mode (CPSR[4:0] = 0x15), is added to allow debug exceptions to be handled similarly to other types of Intel® StrongARM® exceptions.

When a debug exception occurs, the processor switches to debug mode and redirects execution to a debug handler, via the reset vector. After the debug handler begins execution, the debugger can communicate with the debug handler to examine or alter processor state or memory through the JTAG interface.

The debug handler can be downloaded and locked directly into the instruction cache through JTAG so external memory is not required to contain debug handler code.

3.6.3.2 Monitor Mode

In monitor mode, debug exceptions are handled like Intel® StrongARM® prefetch aborts or Intel® StrongARM® data aborts, depending on the cause of the exception.

When a debug exception occurs, the processor switches to abort mode and branches to a debug handler using the pre-fetch abort vector or data abort vector. The debugger then communicates with the debug handler to access processor state or memory contents.

3.6.4 Debug Control and Status Register (DCSR)

The DCSR register is the main control register for the debug unit. [Table 35](#) shows the format of the register. The DCSR register can be accessed in privileged modes by software running on the core or by a debugger through the JTAG interface. Refer to "[SELDCSR JTAG Register](#)" on [page 125](#) for details about accessing DCSR through JTAG.

**Table 35. Debug Control and Status Register (DCSR)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GE	H							TF	TI		TD	TA	TS	TU	TR											SA	MOE	M	E		

Bits	Access	Description	Reset Value	TRST Value
31	SW Read / Write JTAG Read-Only	Global Enable (GE) 0: disables all debug functionality 1: enables all debug functionality	0	unchanged
30	SW Read Only JTAG Read / Write	Halt Mode (H) 0: Monitor Mode 1: Halt Mode	unchanged	0
29:24	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
23	SW Read Only JTAG Read / Write	Trap FIQ (TF)	unchanged	0
22	SW Read Only JTAG Read / Write	Trap IRQ (TI)	unchanged	0
21	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
20	SW Read Only JTAG Read / Write	Trap Data Abort (TD)	unchanged	0
19	SW Read Only JTAG Read / Write	Trap Prefetch Abort (TA)	unchanged	0
18	SW Read Only JTAG Read / Write	Trap Software Interrupt (TS)	unchanged	0
17	SW Read Only JTAG Read / Write	Trap Undefined Instruction (TU)	unchanged	0
16	SW Read Only JTAG Read / Write	Trap Reset (TR)	unchanged	0
15:6	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
5	SW Read / Write JTAG Read-Only	Sticky Abort (SA)	0	unchanged
4:2	SW Read / Write JTAG Read-Only	Method Of Entry (MOE) 000: Processor Reset 001: Instruction Breakpoint Hit 010: Data Breakpoint Hit 011: BKPT Instruction Executed 100: External Debug Event Asserted 101: Vector Trap Occurred 110: Trace Buffer Full Break 111: Reserved	0b000	unchanged
1	SW Read / Write JTAG Read-Only	Trace Buffer Mode (M) 0: Wrap around mode 1: fill-once mode	0	unchanged
0	SW Read / Write JTAG Read-Only	Trace Buffer Enable (E) 0: Disabled 1: Enabled	0	unchanged



3.6.4.1 Global Enable Bit (GE)

The Global Enable bit disables and enables all debug functionality (except the reset vector trap). Following a processor reset, this bit is clear so all debug functionality is disabled. When debug functionality is disabled, the BKPT instruction becomes a loop and external debug breaks, hardware breakpoints, and non-reset vector traps are ignored.

3.6.4.2 Halt Mode Bit (H)

The Halt Mode bit configures the debug unit for either halt mode or monitor mode.

3.6.4.3 Vector Trap Bits (TF, TI, TD, TA, TS, TU, TR)

The Vector Trap bits allow instruction breakpoints to be set on exception vectors without using up any of the breakpoint registers. When a bit is set, it acts as if an instruction breakpoint was set up on the corresponding exception vector. A debug exception is generated before the instruction in the exception vector executes.

Software running on the IXP45X/IXP46X network processors must set the Global Enable bit and the debugger must set the Halt Mode bit and the appropriate vector trap bit through JTAG to set up a non-reset vector trap.

To set up a reset vector trap, the debugger sets the Halt Mode bit and reset vector trap bit through JTAG. The Global Enable bit does not effect the reset vector trap. A reset vector trap can be set up before or during a processor reset. When processor reset is de-asserted, a debug exception occurs before the instruction in the reset vector executes.

3.6.4.4 Sticky Abort Bit (SA)

The Sticky Abort bit is only valid in Halt mode. It indicates a data abort occurred within the Special Debug State (see “Halt Mode” on page 115). Since Special Debug State disables all exceptions, a data abort exception does not occur. However, the processor sets the Sticky Abort bit to indicate a data abort was detected. The debugger can use this bit to determine if a data abort was detected during the Special Debug State. The sticky abort bit must be cleared by the debug handler before exiting the debug handler.

3.6.4.5 Method of Entry Bits (MOE)

The Method of Entry bits specify the cause of the most recent debug exception. When multiple exceptions occur in parallel, the processor places the highest priority exception (based on the priorities in Table 36) in the MOE field.

3.6.4.6 Trace Buffer Mode Bit (M)

The Trace Buffer Mode bit selects one of two trace buffer modes:

- Wrap-around mode — Trace buffer fills up and wraps around until a debug exception occurs.
- Fill-once mode — The trace buffer automatically generates a debug exception (trace buffer full break) when it becomes full.

3.6.4.7 Trace Buffer Enable Bit (E)

The Trace Buffer Enable bit enables and disables the trace buffer. Both DCSR.e and DCSR.ge must be set to enable the trace buffer. The processor automatically clears this bit to disable the trace buffer when a debug exception occurs. For more details on the trace buffer refer to “Trace Buffer” on page 132.



3.6.5 Debug Exceptions

A debug exception causes the processor to re-direct execution to a debug event handling routine. The debug architecture of the IXP45X/IXP46X network processors defines the following debug exceptions:

- Instruction breakpoint
- Data breakpoint
- Software breakpoint
- External debug break
- Exception vector trap
- Trace-buffer full break

When a debug exception occurs, the processor's actions depend on whether the debug unit is configured for Halt mode or Monitor mode.

Table 36 shows the priority of debug exceptions relative to other processor exceptions.

Table 36. Event Priority

Event	Priority
Reset	1
Vector Trap	2
Data abort (precise)	3
Data bkpt	4
Data abort (imprecise)	5
External debug break, trace-buffer full	6
FIQ	7
IRQ	8
Instruction breakpoint	9
Pre-fetch abort	10
Undef, SWI, software Bkpt	11

3.6.5.1 Halt Mode

The debugger turns on Halt mode through the JTAG interface by scanning in a value that sets the bit in DCSR. The debugger turns off Halt mode through JTAG, either by scanning in a new DCSR value or by a TRST. Processor reset does not effect the value of the Halt mode bit.

When halt mode is active, the processor uses the reset vector as the debug vector. The debug handler and exception vectors can be downloaded directly into the instruction cache, to intercept the default vectors and reset handler, or they can be resident in external memory. Downloading into the instruction cache allows a system with memory problems, or no external memory, to be debugged. Refer to [“Downloading Code in ICache” on page 139](#) for details about downloading code into the instruction cache.

During Halt mode, software running on the IXP45X/IXP46X network processors cannot access DCSR, or any of hardware breakpoint registers, unless the processor is in Special Debug State (SDS), described below.

When a debug exception occurs during Halt mode, the processor takes the following actions:



- Disables the trace buffer
- Sets DCSR.moe encoding
- Processor enters a Special Debug State (SDS)
- For data breakpoints, trace buffer full break, and external debug break:
R14_dbg = PC of the next instruction to execute + 4
for instruction breakpoints and software breakpoints and vector traps:
R14_dbg = PC of the aborted instruction + 4
- SPSR_dbg = CPSR
- CPSR[4:0] = 0b10101 (DEBUG mode)
- CPSR[5] = 0
- CPSR[6] = 1
- CPSR[7] = 1
- PC = 0x0

Note: When the vector table is relocated (CP15 Control Register[13] = 1), the debug vector is relocated to 0xffff0000.

Following a debug exception, the processor switches to debug mode and enters SDS, which allows the following special functionality:

- All events are disabled. SWI or undefined instructions have unpredictable results. The processor ignores pre-fetch aborts, FIQ and IRQ (SDS disables FIQ and IRQ regardless of the enable values in the CPSR). The processor reports data aborts detected during SDS by setting the Sticky Abort bit in the DCSR, but does not generate an exception (processor also sets up FSR and FAR as it normally would for a data abort).
- Normally, during halt mode, software cannot write the hardware breakpoint registers or the DCSR. However, during the SDS, software has write access to the breakpoint registers (see [“HW Breakpoint Resources” on page 117](#)) and the DCSR (see [Table 35, “Debug Control and Status Register \(DCSR\)” on page 113](#)).
- The IMMU is disabled. In halt mode, since the debug handler would typically be downloaded directly into the IC, it would not be appropriate to do TLB accesses or translation walks, since there may not be any external memory or if there is, the translation table or TLB may not contain a valid mapping for the debug handler code. To avoid these problems, the processor internally disables the IMMU during SDS.
- The PID is disabled for instruction fetches. This prevents fetches of the debug handler code from being remapped to a different address than where the code was downloaded.

The SDS remains in effect regardless of the processor mode. This allows the debug handler to switch to other modes, maintaining SDS functionality. Entering user mode may cause unpredictable behavior. The processor exits SDS following a CPSR restore operation.

When exiting, the debug handler should use:

```
subs pc, lr, #4
```

This restores CPSR, turns off all of SDS functionality, and branches to the target instruction.



3.6.5.2 Monitor Mode

In monitor mode, the processor handles debug exceptions like normal Intel® StrongARM® exceptions. If debug functionality is enabled (DCSR[31] = 1) and the processor is in Monitor mode, debug exceptions cause either a data abort or a pre-fetch abort.

The following debug exceptions cause data aborts:

- Data breakpoint
- External debug break
- Trace-buffer full break

The following debug exceptions cause pre-fetch aborts:

- Instruction breakpoint
- BKPT instruction

The processor ignores vector traps during monitor mode.

When an exception occurs in monitor mode, the processor takes the following actions:

- Disables the trace buffer
- Sets DCSR.moe encoding
- Sets FSR[9]
- R14_abt = PC of the next instruction to execute + 4 (for Data Aborts)
R14_abt = PC of the faulting instruction + 4 (for Prefetch Aborts)
- SPSR_abt = CPSR
- CPSR[4:0] = 0b10111 (ABORT mode)
- CPSR[5] = 0
- CPSR[6] = unchanged
- CPSR[7] = 1
- PC = 0xc (for Prefetch Aborts),
PC = 0x10 (for Data Aborts)

During abort mode, external debug breaks and trace buffer full breaks are internally pended. When the processor exits abort mode, either through a CPSR restore or a write directly to the CPSR, the pended debug breaks will immediately generate a debug exception. Any pending debug breaks are cleared out when any type of debug exception occurs.

When exiting, the debug handler should do a CPSR restore operation that branches to the next instruction to be executed in the program under debug.

3.6.6 HW Breakpoint Resources

The debug architecture of the IXP45X/IXP46X network processors defines two instruction and two data breakpoint registers, denoted IBCR0, IBCR1, DBR0, and DBR1.

The instruction and data address breakpoint registers are 32-bit registers. The instruction breakpoint causes a break before execution of the target instruction. The data breakpoint causes a break after the memory access has been issued.



In this section Modified Virtual Address (MVA) refers to the virtual address ORed with the PID. Refer to “[Register 13: Process ID](#)” on [page 106](#) for more details on the PID. The processor does not OR the PID with the specified breakpoint address prior to doing address comparison. This must be done by the programmer and written to the breakpoint register as the MVA. This applies to data and instruction breakpoints.

3.6.6.1 Instruction Breakpoints

The Debug architecture defines two instruction breakpoint registers (IBCR0 and IBCR1). The format of these registers is shown in Table 37., Instruction Breakpoint Address and Control Register (IBCRx). In Intel® StrongARM® mode, the upper 30 bits contain a word aligned MVA to break on. In Thumb mode, the upper 31 bits contain a half-word aligned MVA to break on. In both modes, bit 0 enables and disables that instruction breakpoint register. Enabling instruction breakpoints while debug is globally disabled (DCSR.GE=0) may result in unpredictable behavior.

Table 37. Instruction Breakpoint Address and Control Register (IBCRx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBCRx																															E
reset value: unpredictable address, disabled																															
Bits	Access										Description																				
31:1	Read / Write										Instruction Breakpoint MVA in Intel® StrongARM® mode, IBCRx[1] is ignored																				
0	Read / Write										IBCRx Enable (E) - 0 = Breakpoint disabled 1 = Breakpoint enabled																				

An instruction breakpoint will generate a debug exception before the instruction at the address specified in the IBCR executes. When an instruction breakpoint occurs, the processor sets the DBCR.moe bits to 0b001.

Software must disable the breakpoint before exiting the handler. This allows the breakpointed instruction to execute after the exception is handled.

Single step execution is accomplished using the instruction breakpoint registers and must be completely handled in software (either on the host or by the debug handler).

3.6.6.2 Data Breakpoints

The debug architecture of the IXP45X/IXP46X network processors defines two data breakpoint registers (DBR0, DBR1). The format of the registers is shown in [Table 38](#).

Table 38. Data Breakpoint Register (DBRx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBRx																															
reset value: unpredictable																															
Bits	Access										Description																				
31:0	Read / Write										DBR0 : Data Breakpoint MVA DBR1 : Data Address Mask OR Data Breakpoint MVA																				



DBR0 is a dedicated data address breakpoint register. DBR1 can be programmed for one of two operations:

- Data address mask
- Second data address breakpoint

The DBCON register controls the functionality of DBR1, as well as the enables for both DBRs. DBCON also controls what type of memory access to break on.

Table 39. Data Breakpoint Controls Register (DBCON)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								M						E1	E0
reset value: 0x00000000																															
Bits		Access		Description																											
31:9		Read-as-Zero / Write-ignored		Reserved																											
8		Read / Write		DBR1 Mode (M) - 0: DBR1 = Data Address Breakpoint 1: DBR1 = Data Address Mask																											
7:4		Read-as-Zero / Write-ignored		Reserved																											
3:2		Read / Write		DBR1 Enable (E1) - When DBR1 = Data Address Breakpoint 0b00: DBR1 disabled 0b01: DBR1 enabled, Store only 0b10: DBR1 enabled, Any data access, load or store 0b11: DBR1 enabled, Load only When DBR1 = Data Address Mask this field has no effect																											
1:0		Read / Write		DBR0 Enable (E0) - 0b00: DBR0 disabled 0b01: DBR0 enabled, Store only 0b10: DBR0 enabled, Any data access, load or store 0b11: DBR0 enabled, Load only																											

When DBR1 is programmed as a data address mask, it is used in conjunction with the address in DBR0. The bits set in DBR1 are ignored by the processor when comparing the address of a memory access with the address in DBR0. Using DBR1 as a data address mask allows a range of addresses to generate a data breakpoint. When DBR1 is selected as a data address mask, it is unaffected by the E1 field of DBCON. The mask is used only when DBR0 is enabled.

When DBR1 is programmed as a second data address breakpoint, it functions independently of DBR0. In this case, the DBCON.E1 controls DBR1.

A data breakpoint is triggered if the memory access matches the access type and the address of any byte within the memory access matches the address in DBRx. For example, LDR triggers a breakpoint if DBCON.E0 is 0b10 or 0b11, and the address of any of the 4 bytes accessed by the load matches the address in DBR0.

The processor does not trigger data breakpoints for the PLD instruction or any CP15, register 7, 8, 9, or 10 functions. Any other type of memory access can trigger a data breakpoint. For data breakpoint purposes the SWP and SWPB instructions are treated as stores - they will not cause a data breakpoint if the breakpoint is set up to break on loads only and an address match occurs.

On unaligned memory accesses, breakpoint address comparison is done on a word-aligned address (aligned down to word boundary).



When a memory access triggers a data breakpoint, the breakpoint is reported after the access is issued. The memory access will not be aborted by the processor. The actual timing of when the access completes with respect to the start of the debug handler depends on the memory configuration.

On a data breakpoint, the processor generates a debug exception and re-directs execution to the debug handler before the next instruction executes. The processor reports the data breakpoint by setting the DCSR.MOE to 0b010. The link register of a data breakpoint is always PC (of the next instruction to execute) + 4, regardless of whether the processor is configured for monitor mode or halt mode.

3.6.7 Software Breakpoints

Mnemonics: BKPT (See *ARM* Architecture Reference Manual*, ARMv5T)

Operation: If DCSR[31] = 0, BKPT is a nop;
If DCSR[31] = 1, BKPT causes a debug exception

The processor handles the software breakpoint as described in [“Debug Exceptions” on page 115](#).

3.6.8 Transmit/Receive Control Register

Communications between the debug handler and debugger are controlled through handshaking bits that ensures the debugger and debug handler make synchronized accesses to TX and RX. The debugger side of the handshaking is accessed through the DBGTX ([“DBGTX JTAG Register” on page 127](#)) and DBGRX ([“DBGRX JTAG Register” on page 128](#)) JTAG Data Registers, depending on the direction of the data transfer. The debug handler uses separate handshaking bits in TXRXCTRL register for accessing TX and RX.

The TXRXCTRL register also contains two other bits that support high-speed download. One bit indicates an overflow condition that occurs when the debugger attempts to write the RX register before the debug handler has read the previous data written to RX. The other bit is used by the debug handler as a branch flag during high-speed download.

All of the bits in the TXRXCTRL register are placed such that they can be read directly into the CC flags in the CPSR with an MRC (with Rd = PC). The subsequent instruction can then conditionally execute based on the updated CC value

**Table 40. TX RX Control Register (TXRXCTRL)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RR	OV	D	TR																												

reset value: 0x00000000

Bits	Access	Description
31	SW Read-only / Write-ignored JTAG Write-only	RR RX Register Ready
30	SW Read / Write	OV RX overflow sticky flag
29	SW Read-only/ Write-ignored JTAG Write-only	D High-speed download flag
28	SW Read-only/ Write-ignored JTAG Write-only	TR TX Register Ready
27:0	Read-as-Zero / Write-ignored	Reserved

3.6.8.1 RX Register Ready Bit (RR)

The debugger and debug handler use the RR bit to synchronize accesses to RX. Normally, the debugger and debug handler use a handshaking scheme that requires both sides to poll the RR bit. To support higher download performance for large amounts of data, a high-speed download handshaking scheme can be used in which only the debug handler polls the RR bit before accessing the RX register, while the debugger continuously downloads data.

Table 41 shows the normal handshaking used to access the RX register.

Table 41. Normal RX Handshaking

Debugger Actions
- Debugger wants to send data to debug handler. - Before writing new data to the RX register, the debugger polls RR through JTAG until the bit is cleared. - After the debugger reads a '0' from the RR bit, it scans data into JTAG to write to the RX register and sets the valid bit. The write to the RX register automatically sets the RR bit.
Debug Handler Actions
- Debug handler is expecting data from the debugger. - The debug handler polls the RR bit until it is set, indicating data in the RX register is valid. - Once the RR bit is set, the debug handler reads the new data from the RX register. The read operation automatically clears the RR bit.

When data is being downloaded by the debugger, part of the normal handshaking can be bypassed to allow the download rate to be increased. Table 42 shows the handshaking used when the debugger is doing a high-speed download. Note that before the high-speed download can start, both the debugger and debug handler must be synchronized, such that the debug handler is executing a routine that supports the high-speed download.

Although it is similar to the normal handshaking, the debugger polling of RR is bypassed with the assumption that the debug handler can read the previous data from RX before the debugger can scan in the new data.



Table 42. High-Speed Download Handshaking States

Debugger Actions
• Debugger wants to transfer code into the IXP45X/IXP46X network processors' system memory. • Prior to starting download, the debugger must poll RR bit until it is clear. Once the RR bit is clear, indicating the debug handler is ready, the debugger starts the download. • The debugger scans data into JTAG to write to the RX register with the download bit and the valid bit set. Following the write to RX, the RR bit and D bit are automatically set in TXRXCTRL. • Without polling of RR to see whether the debug handler has read the data just scanned in, the debugger continues scanning in new data into JTAG for RX, with the download bit and the valid bit set. • An overflow condition occurs if the debug handler does not read the previous data before the debugger completes scanning in the new data, (See "Overflow Flag (OV)" on page 122 for more details on the overflow condition). • After completing the download, the debugger clears the D bit allowing the debug handler to exit the download loop.
Debug Handler Actions
• Debug is handler in a routine waiting to write data out to memory. The routine loops based on the D bit in TXRXCTRL. • The debug handler polls the RR bit until it is set. It then reads the Rx register, and writes it out to memory. The handler loops, repeating these operations until the debugger clears the D bit.

3.6.8.2 Overflow Flag (OV)

The Overflow flag is a sticky flag that is set when the debugger writes to the RX register while the RR bit is set.

The flag is used during high-speed download to indicate that some data was lost. The assumption during high-speed download is that the time it takes for the debugger to shift in the next data word is greater than the time necessary for the debug handler to process the previous data word. So, before the debugger shifts in the next data word, the handler will be polling for that data.

However, if the handler incurs stalls that are long enough such that the handler is still processing the previous data when the debugger completes shifting in the next data word, an overflow condition occurs and the OV bit is set.

Once set, the overflow flag will remain set, until cleared by a write to TXRXCTRL with an MCR. After the debugger completes the download, it can examine the OV bit to determine if an overflow occurred. The debug handler software is responsible for saving the address of the last valid store before the overflow occurred.

3.6.8.3 Download Flag (D)

The value of the download flag is set by the debugger through JTAG. This flag is used during high-speed download to replace a loop counter.

The download flag becomes especially useful when an overflow occurs. If a loop counter is used, and an overflow occurs, the debug handler cannot determine how many data words overflowed. Therefore the debug handler counter may get out of sync with the debugger — the debugger may finish downloading the data, but the debug handler counter may indicate there is more data to be downloaded - this may result in unpredictable behavior of the debug handler.

Using the download flag, the debug handler loops until the debugger clears the flag. Therefore, when doing a high-speed download, for each data word downloaded, the debugger should set the D bit.



3.6.8.4 TX Register Ready Bit (TR)

The debugger and debug handler use the TR bit to synchronize accesses to the TX register. The debugger and debug handler must poll the TR bit before accessing the TX register. Table 43 shows the handshaking used to access the TX register.

Table 43. TX Handshaking

Debugger Actions
- Debugger is expecting data from the debug handler. - Before reading data from the TX register, the debugger polls the TR bit through JTAG until the bit is set. NOTE: while polling TR, the debugger must scan out the TR bit and the TX register data. - Reading a '1' from the TR bit, indicates that the TX data scanned out is valid - The action of scanning out data when the TR bit is set, automatically clears TR.
Debug Handler Actions
- Debug handler wants to send data to the debugger (in response to a previous request). - The debug handler polls the TR bit to determine when the TX register is empty (any previous data has been read out by the debugger). The handler polls the TR bit until it is clear. - Once the TR bit is clear, the debug handler writes new data to the TX register. The write operation automatically sets the TR bit.

3.6.8.5 Conditional Execution Using TXRXCTRL

All of the bits in TXRXCTRL are placed such that they can be read directly into the CC flags using an MCR instruction. To simplify the debug handler, the TXRXCTRL register should be read using the following instruction:

```
mrc p14, 0, r15, C14, C0, 0
```

This instruction will directly update the condition codes in the CPSR. The debug handler can then conditionally execute based on each C bit. Table 44 shows the mnemonic extension to conditionally execute based on whether the TXRXCTRL bit is set or clear.

Table 44. TXRXCTRL Mnemonic Extensions

TXRXCTRL bit	Mnemonic Extension to Execute If Bit Set	Mnemonic Extension to Execute If Bit Clear
31 (to N flag)	MI	PL
30 (to Z flag)	EQ	NE
29 (to C flag)	CS	CC
28 (to V flag)	VS	VC

The following example is a code sequence in which the debug handler polls the TXRXCTRL handshaking bit to determine when the debugger has completed its write to RX and the data is ready for the debug handler to read.

```
loop: mcr    p14, 0, r15, c14, c0, 0 # read the handshaking bit in TXRXCTRL
      mcrmi  p14, 0, r0, c9, c0, 0 # if RX is valid, read it
      bpl    loop                    # if RX is not valid, loop
```



3.6.9 Transmit Register

The TX register is the debug handler transmit buffer. The debug handler sends data to the debugger through this register.

Table 45. TX Register

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0																															
TX																															
reset value: unpredictable																															
Bits	Access										Description																				
31:0	SW Read / Write JTAG Read-only										Debug handler writes data to send to debugger																				

Since the TX register is accessed by the debug handler (using MCR/MRC) and the debugger (through JTAG), handshaking is required to prevent the debug handler from writing new data before the debugger reads the previous data.

The TX register handshaking is described in [Table 43, “TX Handshaking” on page 123](#).

3.6.10 Receive Register

The RX register is the receive buffer used by the debug handler to get data sent by the debugger through the JTAG interface.

Table 46. RX Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX																															
reset value: unpredictable																															
Bits		Access										Description																			
31:0		SW Read-only JTAG Write-only										Software reads to receives data/commands from debugger																			

Since the RX register is accessed by the debug handler (using MRC) and the debugger (through JTAG), handshaking is required to prevent the debugger from writing new data to the register before the debug handler reads the previous data out. The handshaking is described in [“RX Register Ready Bit \(RR\)” on page 121](#).

3.6.11 Debug JTAG Access

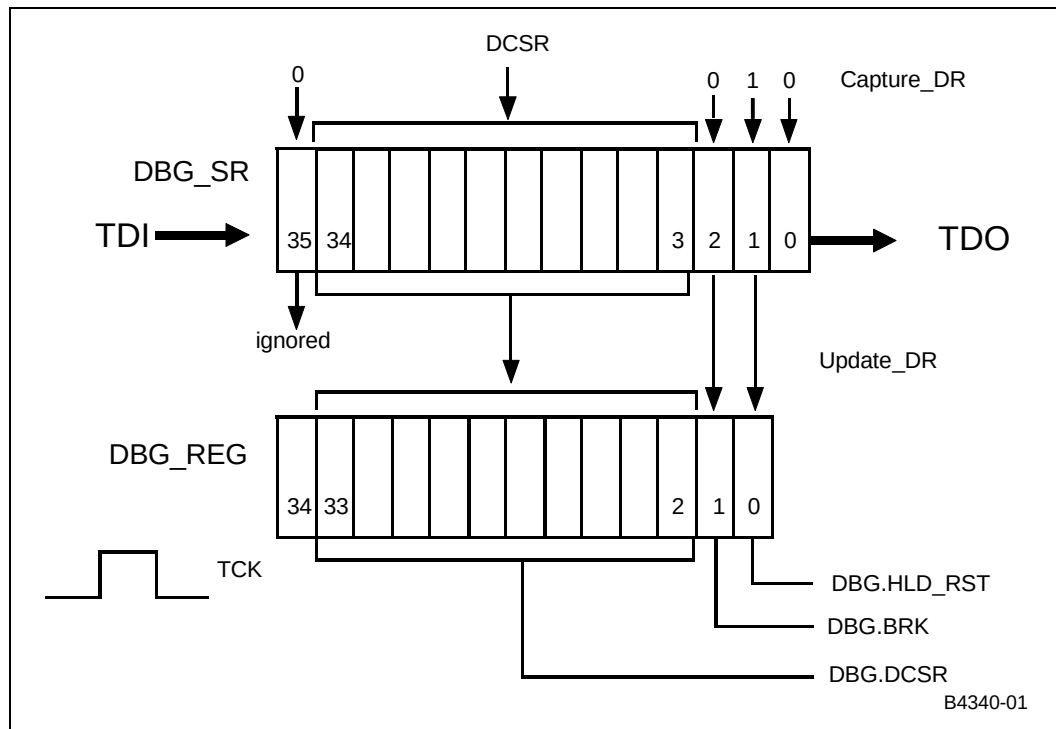
There are four JTAG instructions used by the debugger during software debug: LDIC, SELDCSR, DBGTX and DBGRX. LDIC is described in [“Downloading Code in ICache” on page 139](#). The other three JTAG instructions are described in this section.

SELDCSR, DBGTX and DBGRX use a common 36-bit shift register (DBG_SR). New data is shifted in and captured data out through the DBG_SR. In the UPDATE_DR state, the new data shifted into the appropriate data register.

Update_DR parallel loads the new DCSR value into DBG_REG[33:2]. This value is then loaded into the actual DCSR register. All bits defined as JTAG writable in [Table 35](#), “Debug Control and Status Register (DCSR)” on [page 113](#) are updated.

An external host and the debug handler running on the IXP45X/IXP46X network processors must synchronize access the DCSR. If one side writes the DCSR at the same side the other side reads the DCSR, the results are unpredictable.

Figure 14. SELDCSR Data Register



3.6.11.2.1 DBG.HLD_RST

The debugger uses DBG.HLD_RST when loading code into the instruction cache during a processor reset. Details about loading code into the instruction cache are in [“Downloading Code in ICache” on page 139](#).

The debugger must set DBG.HLD_RST before or during assertion of the reset pin. Once DBG.HLD_RST is set, the reset pin can be de-asserted, and the processor will internally remain in reset. The debugger can then load debug handler code into the instruction cache before the processor begins executing any code.

Once the code download is complete, the debugger must clear DBG.HLD_RST. This takes the processor out of reset, and execution begins at the reset vector.

A debugger sets DBG.HLD_RST in one of two ways:

- Either by taking the JTAG state machine into the Capture_DR state, which automatically loads DBG_SR[1] with ‘1’, then the Exit2 state, followed by the Update_Dr state. This will set the DBG.HLD_RST, clear DBG.BRK, and leave the DCSR unchanged (the DCSR bits captured in DBG_SR[34:3] are written back to the DCSR on the Update_DR).
- Alternatively, a ‘1’ can be scanned into DBG_SR[1], with the appropriate value scanned in for the DCSR and DBG.BRK.



DBG.HLD_RST can only be cleared by scanning in a '0' to DBG_SR[1] and scanning in the appropriate values for the DCSR and DBG.BRK.

3.6.11.2.2 **DBG.BRK**

DBG.BRK allows the debugger to generate an external debug break and asynchronously re-direct execution to a debug handling routine.

A debugger sets an external debug break by scanning data into the DBG_SR with DBG_SR[2] set and the desired value to set the DCSR JTAG writable bits in DBG_SR[34:3].

Once an external debug break is set, it remains set internally until a debug exception occurs. In Monitor mode, external debug breaks detected during abort mode are pended until the processor exits abort mode. In Halt mode, breaks detected during SDS are pended until the processor exits SDS. When an external debug break is detected outside of these two cases, the processor ceases executing instructions as quickly as possible. This minimizes breakpoint skid, by reducing the number of instructions that can execute after the external debug break is requested. However, the processor will continue to process any instructions which may have already begun execution. Debug mode will not be entered until all processor activity has ceased in an orderly fashion.

3.6.11.2.3 **DBG.DCSR**

The DCSR is updated with the value loaded into DBG.DCSR following an Update_DR. Only bits specified as writable by JTAG in [Table 35](#) are updated.

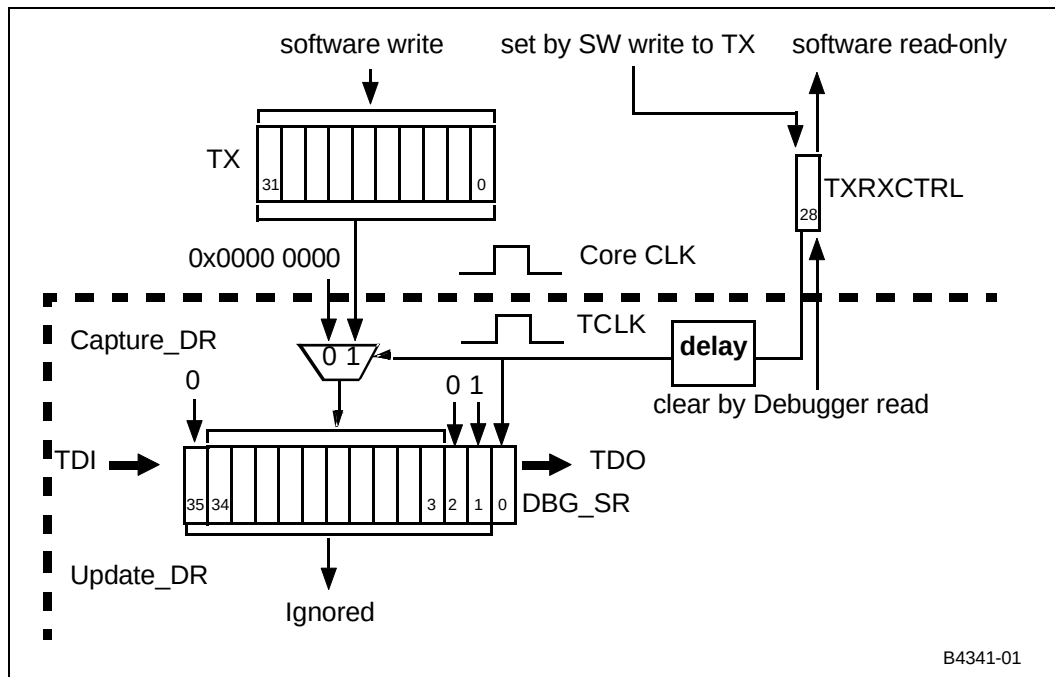
3.6.11.3 **DBGTX JTAG Command**

The 'DBGTX' JTAG instruction selects the DBGTX JTAG data register. The JTAG op code for this instruction is '0b10000'. Once the DBGTX data register is selected, the debugger can receive data from the debug handler.

3.6.11.4 **DBGTX JTAG Register**

The DBGTX JTAG instruction selects the Debug JTAG Data register ([Figure 15](#)). The debugger uses the DBGTX data register to poll for breaks (internal and external) to debug mode and once in debug mode, to read data from the debug handler.

Figure 15. DBGTX Hardware



A Capture_DR loads the TX register value into DBG_SR[34:3] and TXRXCTRL[28] into DBG_SR[0]. The other bits in DBG_SR are loaded as shown in [Figure 35](#).

The captured TX value is scanned out during the Shift_DR state.

Data scanned in is ignored on an Update_DR.

A '1' captured in DBG_SR[0] indicates the captured TX data is valid. After doing a Capture_DR, the debugger must place the JTAG state machine in the Shift_DR state to guarantee that a debugger read clears TXRXCTRL[28].

3.6.11.5 DBGRX JTAG Command

The 'DBGRX' JTAG instruction selects the DBGRX JTAG data register. The JTAG op code for this instruction is '0b00010'. Once the DBGRX data register is selected, the debugger can send data to the debug handler through the RX register.

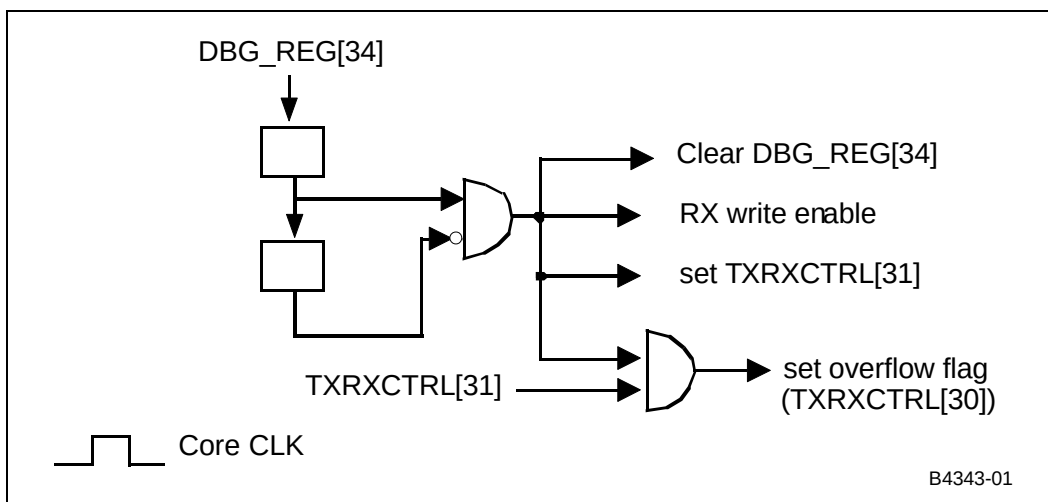
3.6.11.6 DBGRX JTAG Register

The DBGRX JTAG instruction selects the DBGRX JTAG Data register. The debugger uses the DBGRX data register to send data or commands to the debug handler.

and sets DBG_REG[34] to signal the data is valid. Since DBG_REG[34] is never cleared by the debugger in this case, the '0' to '1' transition used to enable the debugger write to RX would not occur.

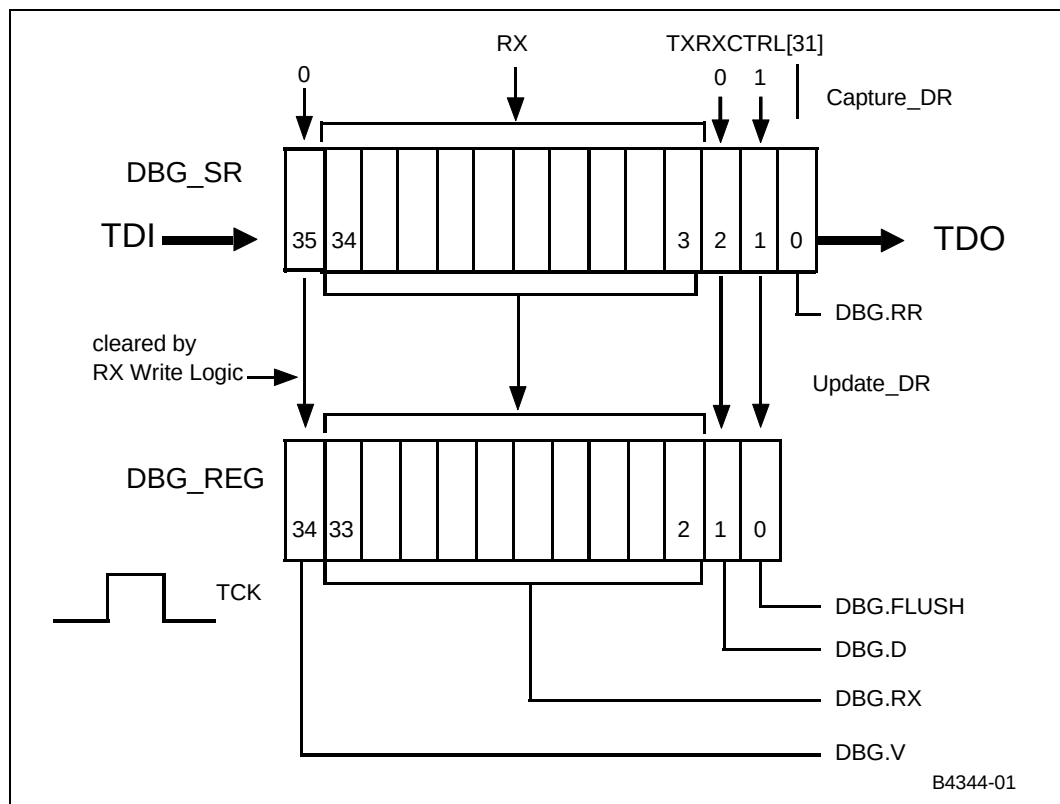
- Set TXRXCTRL[31] - When the debugger writes new data to RX, the logic automatically sets TXRXCTRL[31], signalling to the debug handler that the data is valid.
- Set the overflow flag (TXRXCTRL[30] - During high-speed download, the debugger does not poll to see if the handler has read the previous data. If the debug handler stalls long enough, the debugger may overwrite the previous data before the handler can read it. The logic sets the overflow flag when the previous data has not been read yet, and the debugger has just written new data to RX.

Figure 17. Rx Write Logic



3.6.11.6.2 DBGRX Data Register

The bits in the DBGRX data register (Figure 18) are used by the debugger to send data to the processor. The data register also contains a bit to flush previously written data and a high-speed download flag.

**Figure 18. DBGRX Data Register****3.6.11.6.3 DBG.RR**

The debugger uses **DBG.RR** as part of the synchronization that occurs between the debugger and debug handler for accessing **RX**. This bit contains the value of **TXRXCTRL[31]** after a **Capture_DR**. The debug handler automatically sets **TXRXCTRL[31]** by doing a write to **RX**.

The debugger polls **DBG.RR** to determine when the handler has read the previous data from **RX**.

The debugger sets **TXRXCTRL[31]** by setting the **DBG.V** bit.

3.6.11.6.4 DBG.V

The debugger sets this bit to indicate the data scanned into **DBG_SR[34:3]** is valid data to write to **RX**. **DBG.V** is an input to the **RX Write Logic** and is also cleared by the **RX Write Logic**.

When this bit is set, the data scanned into the **DBG_SR** will be written to **RX** following an **Update_DR**. If **DBG.V** is not set and the debugger does an **Update_DR**, **RX** will be unchanged.

This bit does not affect the actions of **DBG.FLUSH** or **DBG.D**.



3.6.11.6.5 DBG.RX

DBG.RX is written into the RX register based on the output of the RX Write Logic. Any data that needs to be sent from the debugger to the processor must be loaded into DBG.RX with DBG.V set to 1. DBG.RX is loaded from DBG_SR[34:3] when the JTAG enters the Update_DR state.

DBG.RX is written to RX following an Update_DR when the RX Write Logic enables the RX register.

3.6.11.6.6 DBG.D

DBG.D is provided for use during high speed download. This bit is written directly to TXRXCTRL[29]. The debugger sets DBG.D when downloading a block of code or data to the system memory of the IXP45X/IXP46X network processors. The debug handler then uses TXRXCTRL[29] as a branch flag to determine the end of the loop.

Using DBG.D as a branch flags eliminates the need for a loop counter in the debug handler code. This avoids the problem were the debugger's loop counter is out of synchronization with the debug handler's counter because of overflow conditions that may have occurred.

3.6.11.6.7 DBG.FLUSH

DBG.FLUSH allows the debugger to flush any previous data written to RX. Setting DBG.FLUSH clears TXRXCTRL[31].

3.6.11.7 Debug JTAG Data Register Reset Values

Upon asserting TRST, the DEBUG data register is reset. Assertion of the reset pin does not affect the DEBUG data register. Table 47 shows the reset and TRST values for the data register. Note: these values apply for DBG_REG for SELDCSR, DBGTX and DBGRX.

Table 47. DEBUG Data Register Reset Values

Bit	TRST	RESET
DBG_REG[0]	0	unchanged
DBG_REG[1]	0	unchanged
DBG_REG[33:2]	unpredictable	unpredictable
DBG_REG[34]	0	unchanged

3.6.12 Trace Buffer

The 256-entry trace buffer provides the ability to capture control flow information to be used for debugging an application. Two modes are supported:

- The buffer fills up completely and generates a debug exception. Then SW empties the buffer.
- The buffer fills up and wraps around until it is disabled. Then SW empties the buffer.

3.6.12.1 Trace Buffer CP Registers

CP14 defines three registers (see Table 48) for use with the trace buffer. These CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers will cause an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.

**Table 48. CP 14 Trace Buffer Register Summary**

CP14 Register Number	Register Name
11	Trace Buffer Register (TBREG)
12	Checkpoint 0 Register (CHKPT0)
13	Checkpoint 1 Register (CHKPT1)

Any access to the trace buffer registers in User mode will cause an undefined instruction exception. Specifying registers which do not exist has unpredictable results.

3.6.12.1.1 Checkpoint Registers

When the debugger reconstructs a trace history, it is required to start at the oldest trace buffer entry and construct a trace going forward. In fill-once mode and wrap-around mode when the buffer does not wrap around, the trace can be reconstructed by starting from the point in the code where the trace buffer was first enabled.

The difficulty occurs in wrap-around mode when the trace buffer wraps around at least once. In this case the debugger gets a snapshot of the last N control flow changes in the program, where $N \leq \text{size of buffer}$. The debugger does not know the starting address of the oldest entry read from the trace buffer. The checkpoint registers provide reference addresses to help reduce this problem.

Table 49. Checkpoint Register (CHKPTx)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHKPTx																															
reset value: Unpredictable																															
Bits								Access								Description															
31:0								Read/Write								CHKPTx: target address for corresponding entry in trace buffer															

The two checkpoint registers (CHKPT0, CHKPT1) on the IXP45X/IXP46X network processors provide the debugger with two reference addresses to use for re-constructing the trace history.

When the trace buffer is enabled, reading and writing to either checkpoint register has unpredictable results. When the trace buffer is disabled, writing to a checkpoint register sets the register to the value written. Reading the checkpoint registers returns the value of the register.

In normal usage, the checkpoint registers are used to hold target addresses of specific entries in the trace buffer. Only direct and indirect entries get check-pointed. Exception and roll-over messages are never check-pointed. When an entry is check-pointed, the processor sets bit 6 of the message byte to indicate this (refer to [Table 51., Message Byte Formats](#))

When the trace buffer contains only one check-pointed entry, the corresponding checkpoint register is CHKPT0. When the trace buffer wraps around, two entries will typically be check-pointed, usually about half a buffers length apart. In this case, the first (oldest) check-pointed entry read from the trace buffer corresponds to CHKPT1, the second check-pointed entry corresponds to CHKPT0.

Although the checkpoint registers are provided for wrap-around mode, they are still valid in fill-once mode.



3.6.12.1.2 Trace Buffer Register (TBREG)

The trace buffer is read through TBREG, using MRC and MCR. Software should only read the trace buffer when it is disabled. Reading the trace buffer while it is enabled, may cause unpredictable behavior of the trace buffer. Writes to the trace buffer have unpredictable results.

Reading the trace buffer returns the oldest byte in the trace buffer in the least significant byte of TBREG. The byte is either a message byte or one byte of the 32 bit address associated with an indirect branch message. Table 50 shows the format of the trace buffer register.

Table 50. TBREG Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															Data
reset value: unpredictable																															
Bits		Access		Description																											
31:8		Read-as-Zero/Write-ignored		Reserved																											
7:0		Read / Write-unpredictable		Message Byte or Address Byte																											

3.6.13 Trace Buffer Entries

Trace buffer entries consist of either one or five bytes. Most entries are one byte messages indicating the type of control flow change. The target address of the control flow change represented by the message byte is either encoded in the message byte (like for exceptions) or can be determined by looking at the instruction word (like for direct branches). Indirect branches require five bytes per entry. One byte is the message byte identifying it as an indirect branch. The other four bytes make up the target address of the indirect branch. The following sections describe the trace buffer entries in detail.

3.6.13.1 Message Byte

There are two message formats, (exception and non-exception) as shown in Figure 19.

Figure 19. Message Byte Formats

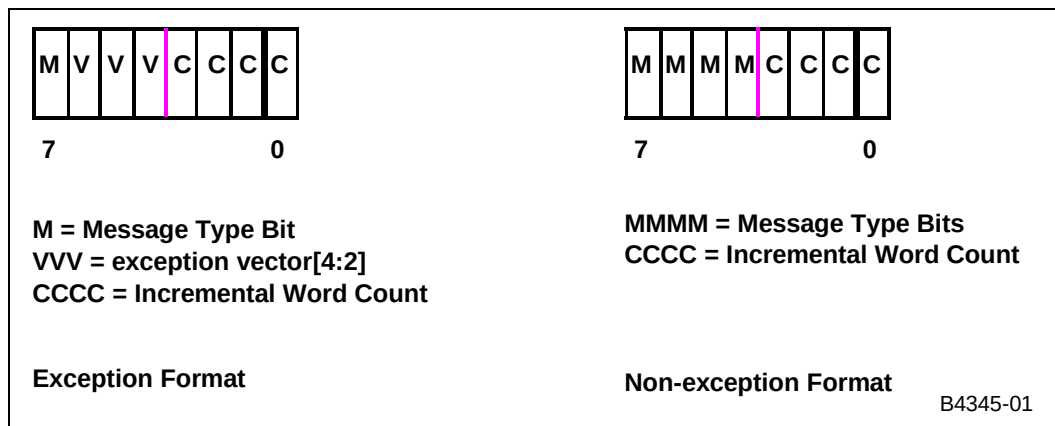




Table 51 shows all of the possible trace messages.

Table 51. Message Byte Formats

Message Name	Message Byte Type	Message Byte Format	# Address Bytes
Exception	exception	0b0VVV CCCC	0
Direct Branch ¹	non-exception	0b1000 CCCC	0
Check-Pointed Direct Branch ¹	non-exception	0b1100 CCCC	0
Indirect Branch ²	non-exception	0b1001 CCCC	4
Check-Pointed Indirect Branch ²	non-exception	0b1101 CCCC	4
Roll-over	non-exception	0b1111 1111	0

Notes:

1. Direct branches include Intel® StrongARM® and THUMB bl, b.
2. Indirect branches include Intel® StrongARM® ldm, ldr, and dproc to PC; Intel® StrongARM® and THUMB bx, blx(1) and blx(2); and THUMB pop.

3.6.13.1.1 Exception Message Byte

When any kind of exception occurs, an exception message is placed in the trace buffer. In an exception message byte, the message type bit (M) is always 0.

The vector exception (VVV) field is used to specify bits[4:2] of the vector address (offset from the base of default or relocated vector table). The vector allows the host SW to identify which exception occurred.

The incremental word count (CCCC) is the instruction count since the last control flow change (not including the current instruction for undef, SWI, and pre-fetch abort). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition of the instruction not matching the CC flags.

A count value of 0 indicates that 0 instructions executed since the last control flow change and the current exception. For example, if a branch is immediate followed by a SWI, a direct branch exception message (for the branch) is followed by an exception message (for the SWI) in the trace buffer. The count value in the exception message will be 0, meaning that 0 instructions executed after the last control flow change (the branch) and before the current control flow change (the SWI). Instead of the SWI, if an IRQ was handled immediately after the branch (before any other instructions executed), the count would still be 0, since no instructions executed after the branch and before the interrupt was handled.

A count of 0b1111 indicates that 15 instructions executed between the last branch and the exception. In this case, an exception was either caused by the 16th instruction (if it is an undefined instruction exception, pre-fetch abort, or SWI) or handled before the 16th instruction executed (for FIQ, IRQ, or data abort).

3.6.13.1.2 Non-Exception Message Byte

Non-exception message bytes are used for direct branches, indirect branches, and rollovers.

In a non-exception message byte, the four-bit message type field (MMMM) specifies the type of message (refer to Table 51).

The incremental word count (CCCC) is the instruction count since the last control flow change (excluding the current branch). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition

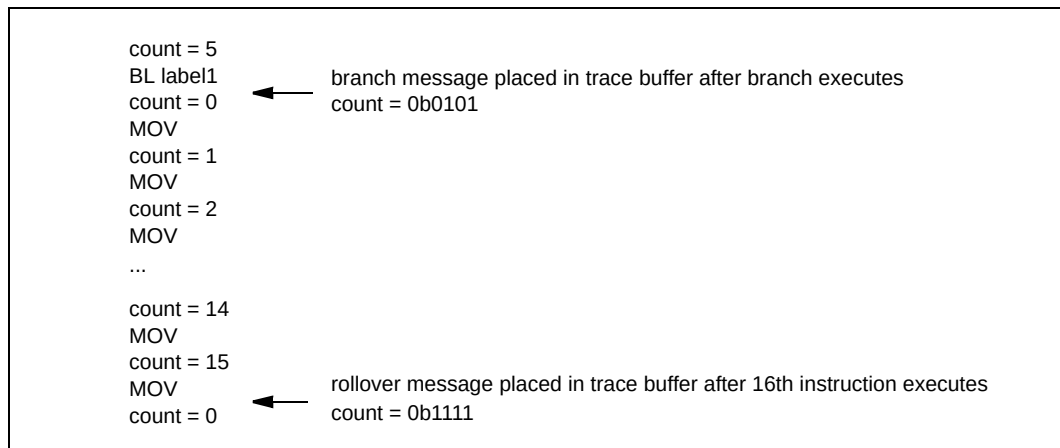


of the instruction not matching the CC flags. In the case of back-to-back branches the word count would be 0 indicating that no instructions executed after the last branch and before the current one.

A rollover message is used to keep track of long traces of code that do not have control flow changes. The rollover message means that 16 instructions have executed since the last message byte was written to the trace buffer.

If the incremental counter reaches its maximum value of 15, a rollover message is written to the trace buffer following the next instruction (which will be the 16th instruction to execute). This is shown in [Example 13](#). The count in the rollover message is 0b1111, indicating that 15 instructions have executed after the last branch and before the current non-branch instruction that caused the rollover message.

Example 13. Rollover Messages Examples



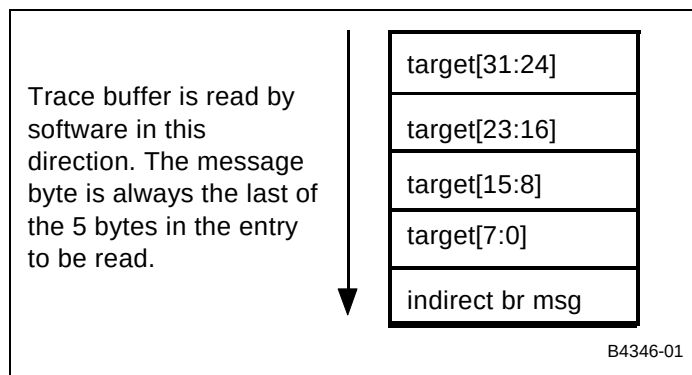
If the 16th instruction is a branch (direct or indirect), the appropriate branch message is placed in the trace buffer instead of the roll-over message. The incremental counter is still set to 0b1111, meaning 15 instructions executed between the last branch and the current branch.

3.6.13.1.3 Address Bytes

Only indirect branch entries contain address bytes in addition to the message byte. Indirect branch entries always have four address bytes indicating the target of that indirect branch. When reading the trace buffer the MSB of the target address is read out first; the LSB is the fourth byte read out; and the indirect branch message byte is the fifth byte read out. The byte organization of the indirect branch message is shown in [Figure 20](#).



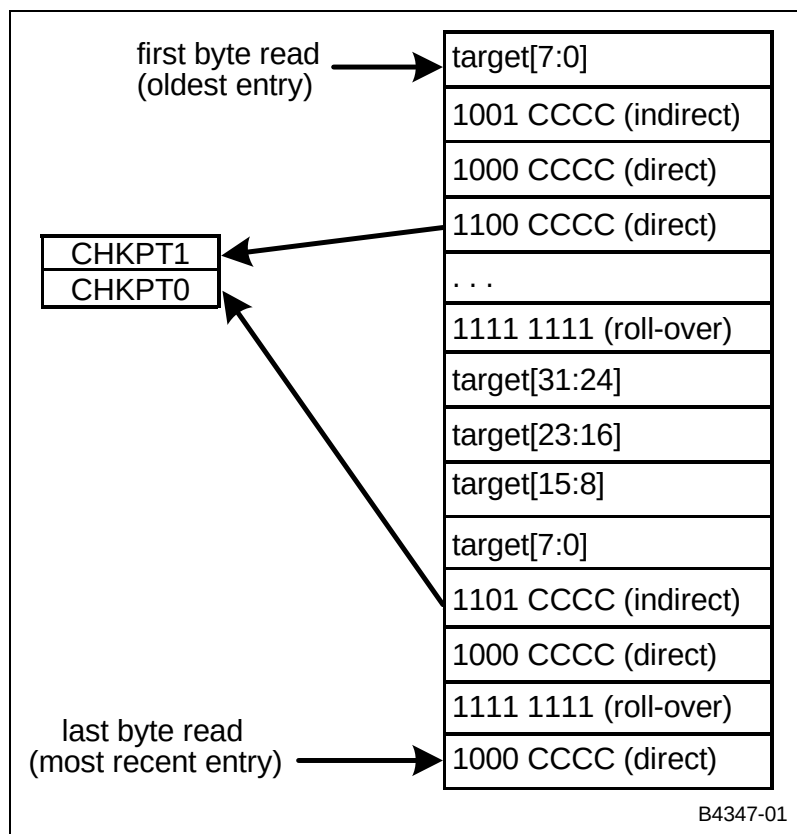
Figure 20. Indirect Branch Entry Address Byte Organization



3.6.13.2 Trace Buffer Usage

The trace buffer for the IXP45X/IXP46X network processors is 256 bytes in length. The first byte read from the buffer represents the oldest trace history information in the buffer. The last (256th) byte read represents the most recent entry in the buffer. The last byte read from the buffer will always be a message byte. This provides the debugger with a starting point for parsing the entries out of the buffer. Because the debugger needs the last byte as a starting point when parsing the buffer, the entire trace buffer must be read (256 bytes on IXP45X/IXP46X network processors) before the buffer can be parsed. Figure 21 is a high level view of the trace buffer.

Figure 21. High-Level View of Trace Buffer





The trace buffer must be initialized prior to its initial usage, then again prior to each subsequent usage. Initialization is done by reading the entire trace buffer. The process of reading the trace buffer also clears it out (all entries are set to 0b0000 0000), so when the trace buffer has been used to capture a trace, the process of reading the captured trace data also re-initializes the trace buffer for its next usage.

The trace buffer can be used to capture a trace up to a processor reset. A processor reset disables the trace buffer, but the contents are unaffected. The trace buffer captures a trace up to the processor reset.

The trace buffer does not capture reset events or debug exceptions.

Since the trace buffer is cleared out before it is used, all entries are initially 0b0000 0000. In fill-once mode, these 0's can be used to identify the first valid entry in the trace buffer. In wrap around mode, in addition to identifying the first valid entry, these 0 entries can be used to determine whether a wrap around occurred.

As the trace buffer is read, the oldest entries are read first. Reading a series of 5 (or more) consecutive "0b0000 0000" entries in the oldest entries indicates that the trace buffer has not wrapped around and the first valid entry will be the first non-zero entry read out.

Reading 4 or less consecutive "0b0000 0000" entries requires a bit more intelligence in the host SW. The host SW must determine whether these 0s are part of the address of an indirect branch message, or whether they are part of the "0b0000 0000" that the trace buffer was initialized with. If the first non-zero message byte is an indirect branch message, then these 0s are part of the address since the address is always read before the indirect branch message (see ["Address Bytes" on page 136](#)). If the first non-zero entry is any other type of message byte, then these 0s indicate that the trace buffer has not wrapped around and that first non-zero entry is the start of the trace.

If the oldest entry from the trace buffer is non-zero, then the trace buffer has either wrapped around or just filled up.

Once the trace buffer has been read and parsed, the host SW should re-create the trace history from oldest trace buffer entry to latest. Trying to re-create the trace going backwards from the latest trace buffer entry may not work in most cases, because once a branch message is encountered, it may not be possible to determine the source of the branch.

In fill-once mode, the return from the debug handler to the application should generate an indirect branch message. The address placed in the trace buffer will be that of the target application instruction. Using this as a starting point, re-creating a trace going forward in time should be straightforward.

In wrap around mode, the host SW should use the checkpoint registers and address bytes from indirect branch entries to re-create the trace going forward. The drawback is that some of the oldest entries in the trace buffer may be untraceable, depending on where the earliest checkpoint (or indirect branch entry) is located. The best case is when the oldest entry in the trace buffer was checkpointed, so the entire trace buffer can be used to re-create the trace. The worst case is when the first checkpoint is in the middle of the trace buffer and no indirect branch messages exist before this checkpoint. In this case, the host SW would have to start at its known address (the first checkpoint) which is half way through the buffer and work forward from there.



3.6.14 Downloading Code in ICache

On the IXP45X/IXP46X network processors, a 2-K mini instruction cache — physically separate from the 32-K main instruction cache — can be used as an on-chip instruction RAM. An external host can download code directly into either instruction cache through JTAG. In addition to downloading code, several cache functions are supported.

Note: A cache line fill from external memory will never be written into the mini-instruction cache. The only way to load a line into the mini-instruction cache is through JTAG.

The IXP45X/IXP46X network processors support loading the instruction cache during reset and during program execution. Loading the instruction cache during normal program execution requires a strict handshaking protocol between software running on the IXP45X/IXP46X network processors and the external host.

In the remainder of this section the term 'instruction cache' applies to either main or mini instruction cache.

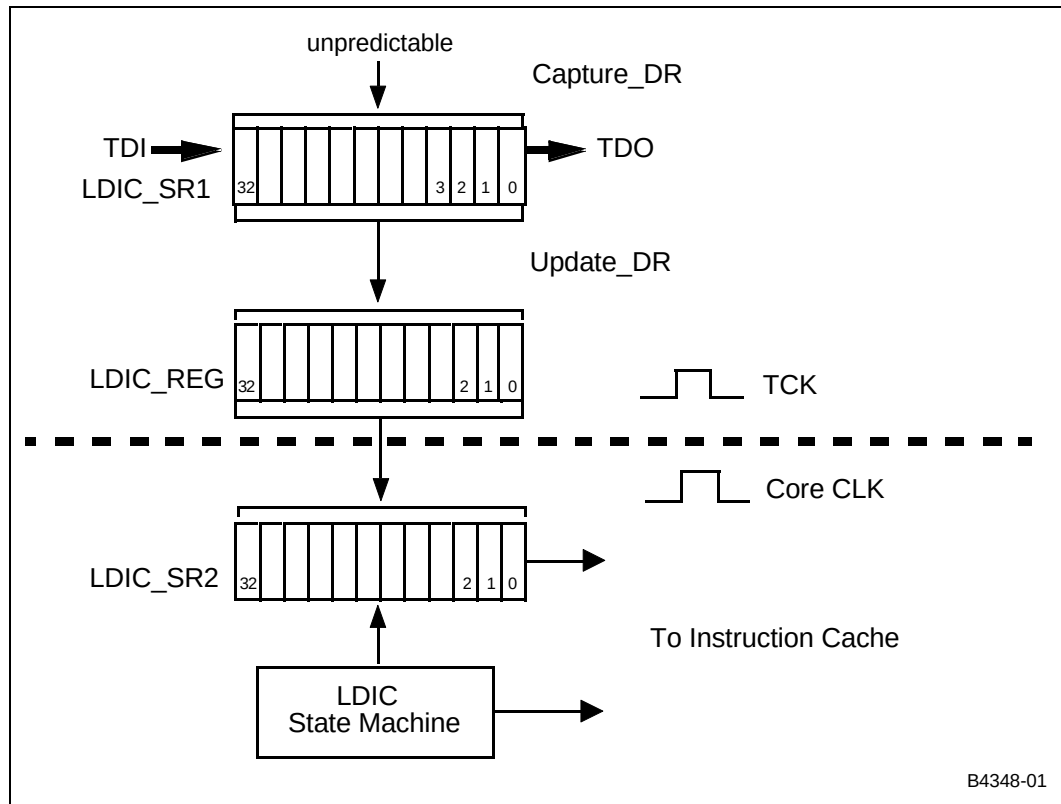
3.6.14.1 LDIC JTAG Command

The LDIC JTAG instruction selects the JTAG data register for loading code into the instruction cache. The JTAG op code for this instruction is '00111'. The LDIC instruction must be in the JTAG instruction register in order to load code directly into the instruction cache through JTAG.

3.6.14.2 LDIC JTAG Data Register

The LDIC JTAG Data Register is selected when the LDIC JTAG instruction is in the JTAG IR. An external host can load and invalidate lines in the instruction cache through this data register.

Figure 22. LDIC JTAG Data Register Hardware



The data loaded into LDIC_SR1 during a Capture_DR is unpredictable.

All LDIC functions and data consists of 33-bit packets which are scanned into LDIC_SR1 during the Shift_DR state.

Update_DR parallel loads LDIC_SR1 into LDIC_REG which is then synchronized with the IXP45X/IXP46X network processors' clock and loaded into the LDIC_SR2. Once data is loaded into LDIC_SR2, the LDIC State Machine turns on and serially shifts the contents of LDIC_SR2 to the instruction cache.

Note that there is a delay from the time of the Update_DR to the time the entire contents of LDIC_SR2 have been shifted to the instruction cache. Removing the LDIC JTAG instruction from the JTAG IR before the entire contents of LDIC_SR2 are sent to the instruction cache, will result in unpredictable behavior. Therefore, following the Update_DR for the last LDIC packet, the LDIC instruction must remain in the JTAG IR for a minimum of 15 TCKs. This ensures the last packet is correctly sent to the instruction cache.



3.6.14.3 LDIC Cache Functions

The IXP45X/IXP46X network processors support four cache functions that can be executed through JTAG. Two functions allow an external host to download code into the main instruction cache or the mini instruction cache through JTAG. Two additional functions are supported to allow lines to be invalidated in the instruction cache. The following table shows the cache functions supported through JTAG.

Table 52. LDIC Cache Functions

Function	Encoding	Arguments	
		Address	# Data Words
Invalidate IC Line	0b000	VA of line to invalidate	0
Invalidate Mini IC	0b001	-	0
Load Main IC	0b010	VA of line to load	8
Load Mini IC	0b011	VA of line to load	8
RESERVED	0b100-0b111	-	-

Invalidate IC line invalidates the line in the instruction cache containing specified virtual address. If the line is not in the cache, the operation has no effect. It does not take any data arguments.

Invalidate Mini IC will invalidate the entire mini instruction cache. It does not effect the main instruction cache. It does not require a virtual address or any data arguments.

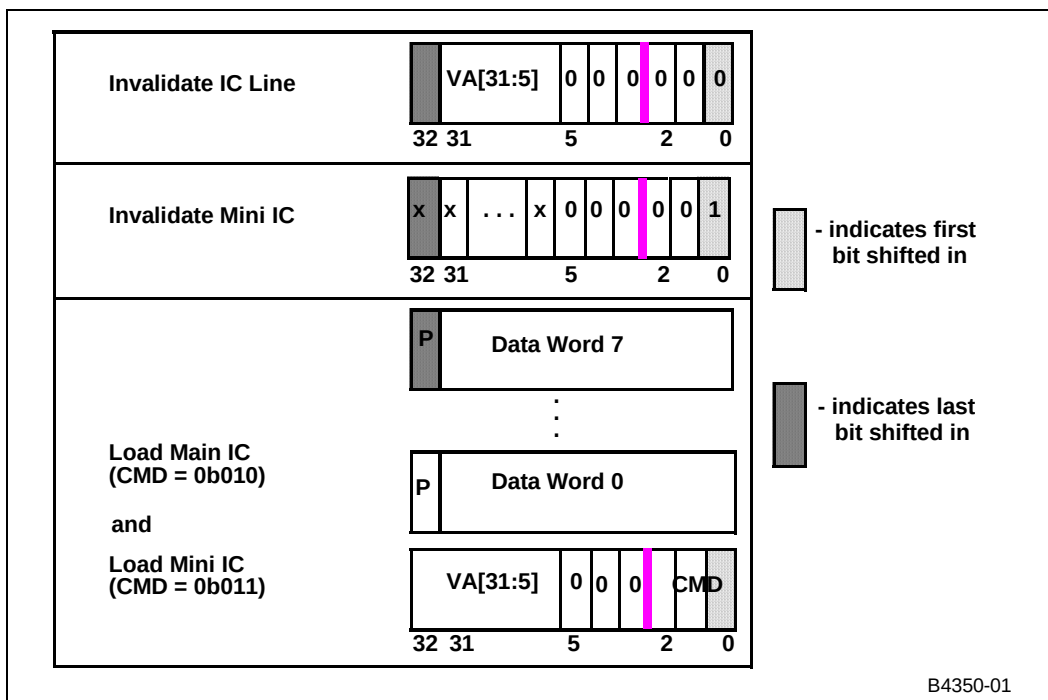
Note:

The LDIC Invalidate Mini IC function does not invalidate the BTB (like the CP15 Invalidate IC function) so software must do this manually where appropriate.

Load Main IC and Load Mini IC write one line of data (eight Intel® StrongARM* instructions) into the specified instruction cache at the specified virtual address.

Each cache function is downloaded through JTAG in 33 bit packets. [Figure 23](#) shows the packet formats for each of the JTAG cache functions. Invalidate IC Line and Invalidate Mini IC each require 1 packet. Load Main IC and Load Mini IC each require 9 packets.

Figure 23. Format of LDIC Cache Functions



All packets are 33 bits in length. Bits [2:0] of the first packet specify the function to execute. For functions that require an address, bits[32:6] of the first packet specify an eight-word aligned address (Packet1[32:6] = VA[31:5]). For Load Main IC and Load Mini IC, eight additional data packets are used to specify eight Intel® StrongARM® instructions to be loaded into the target instruction cache. Bits[31:0] of the data packets contain the data to download. Bit[32] of each data packet is the value of the parity for the data in that packet.

As shown in Figure 23, the first bit shifted in TDI is bit 0 of the first packet. After each 33-bit packet, the host must take the JTAG state machine into the Update_DR state. After the host does an Update_DR and returns the JTAG state machine back to the Shift_DR state, the host can immediately begin shifting in the next 33-bit packet.

3.6.14.4 Loading IC During Reset

Code can be downloaded into the instruction cache through JTAG during a processor reset. This feature is used during software debug to download the debug handler prior to starting an application program. The downloaded handler can then intercept the reset vector and do any necessary setup before the application code executes.

In general, any code downloaded into the instruction cache through JTAG, must be downloaded to addresses that are not already valid in the instruction cache. Failure to meet this requirement will result in unpredictable behavior by the processor. During a processor reset, the instruction cache is typically invalidated, with the exception of the following modes:

- LDIC mode — Active when LDIC JTAG instruction is loaded in the JTAG IR; prevents the mini instruction cache and the main instruction cache from being invalidated during reset.



- **HALT mode** — Active when the Halt Mode bit is set in the DCSR; prevents only the mini instruction cache from being invalidated; main instruction cache is invalidated by reset.

During a cold reset (in which both a processor reset and a JTAG reset occurs) it can be guaranteed that the instruction cache will be invalidated since the JTAG reset takes the processor out of any of the modes listed above.

During a warm reset, if a JTAG reset does not occur, the instruction cache is not invalidated by reset when any of the above modes are active. This situation requires special attention if code needs be downloaded during the warm reset.

Note that while Halt Mode is active, reset can invalidate the main instruction cache. Thus debug handler code downloaded during reset can only be loaded into the mini instruction cache. However, code can be dynamically downloaded into the main instruction cache. (refer to [“Dynamically Loading IC After Reset”](#) on page 147).

The following sections describe the steps necessary to ensure code is correctly downloaded into the instruction cache.

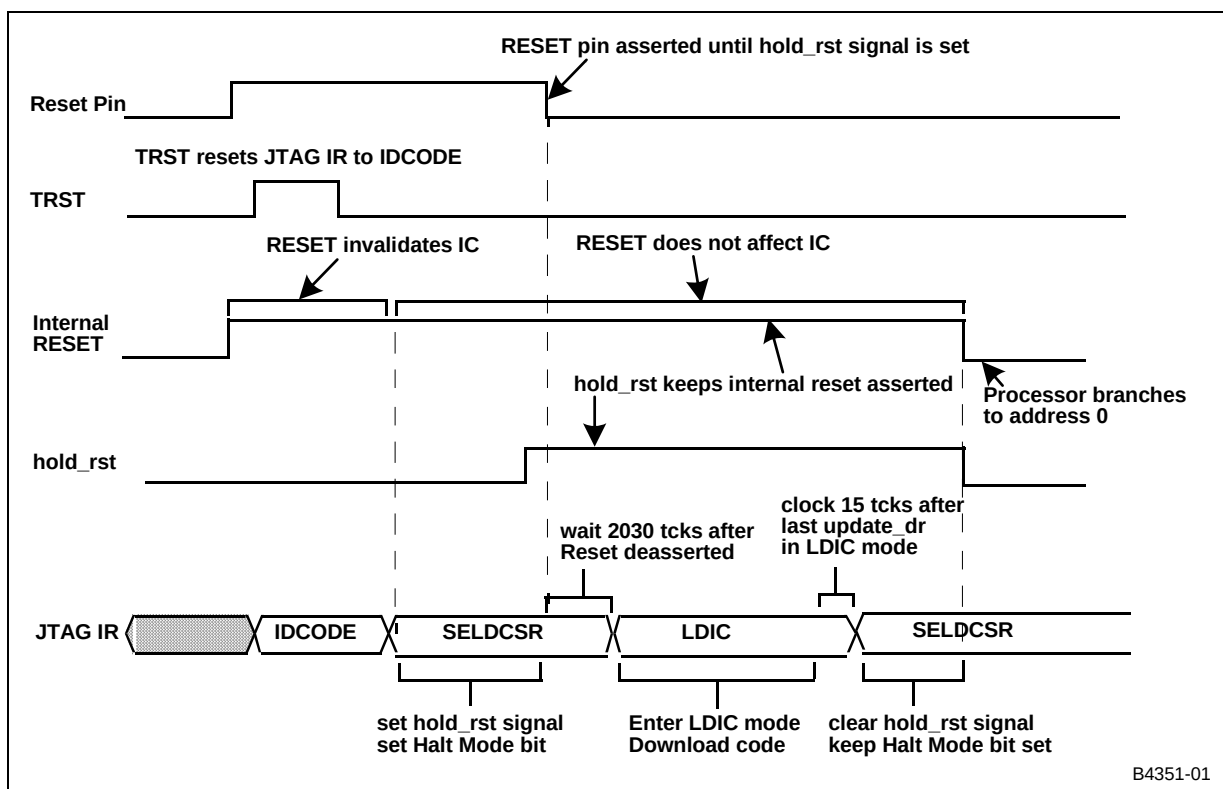
3.6.14.4.1 Loading IC During Cold Reset for Debug

The [Figure 24](#) shows the actions necessary to download code into the instruction cache during a cold reset for debug.

Note: In the [Figure 24](#) hold_rst is a signal that gets set and cleared through JTAG. When the JTAG IR contains the SELDCSR instruction, the hold_rst signal is set to the value scanned into DBG_SR[1].

Note: In the [Figure 24](#) TRST is an internal signal, it is the inverted value of the JTG_TRST_N pin.

Figure 24. Code Download During a Cold Reset For Debug



An external host should take the following steps to load code into the instruction cache following a cold reset:

1. Assert the Reset and JTG_TRST_N pins: This resets the JTAG IR to IDCODE and invalidates the instruction cache (main and mini).
2. Load the SELDCSR JTAG instruction into JTAG IR and scan in a value to set the Halt Mode bit in DCSR and to set the `hold_rst` signal. For details of the SELDCSR, refer to [“SELDCSR JTAG Register” on page 125](#).
3. After `hold_rst` is set, de-assert the Reset pin. Internally the processor remains held in reset.
4. After Reset is de-asserted, wait 2030 TCKs.
5. Load the LDIC JTAG instruction into JTAG IR.
6. Download code into instruction cache in 33-bit packets as described in [“LDIC Cache Functions” on page 141](#).
7. After code download is complete, clock a minimum of 15 TCKs following the last `update_dr` in LDIC mode.
8. Place the SELDCSR JTAG instruction into the JTAG IR and scan in a value to clear the `hold_rst` signal. The Halt Mode bit must remain set to prevent the instruction cache from being invalidated.
9. When `hold_rst` is cleared, internal reset is de-asserted, and the processor executes the reset vector at address 0.



An additional issue for debug is setting up the reset vector trap. This must be done before the internal reset signal is de-asserted. As described in [“Vector Trap Bits \(TF, TI, TD, TA, TS, TU, TR\)” on page 114](#), the Halt Mode and the Trap Reset bits in the DCSR must be set prior to de-asserting reset in order to trap the reset vector. There are two possibilities for setting up the reset vector trap:

- The reset vector trap can be set up before the instruction cache is loaded by scanning in a DCSR value that sets the Trap Reset bit in addition to the Halt Mode bit and the hold_rst signal; OR
- The reset vector trap can be set up after the instruction cache is loaded. In this case, the DCSR should be set up to do a reset vector trap, with the Halt Mode bit and the hold_rst signal remaining set.

In either case, when the debugger clears the hold_rst bit to de-assert internal reset, the debugger must set the Halt Mode and Trap Reset bits in the DCSR.

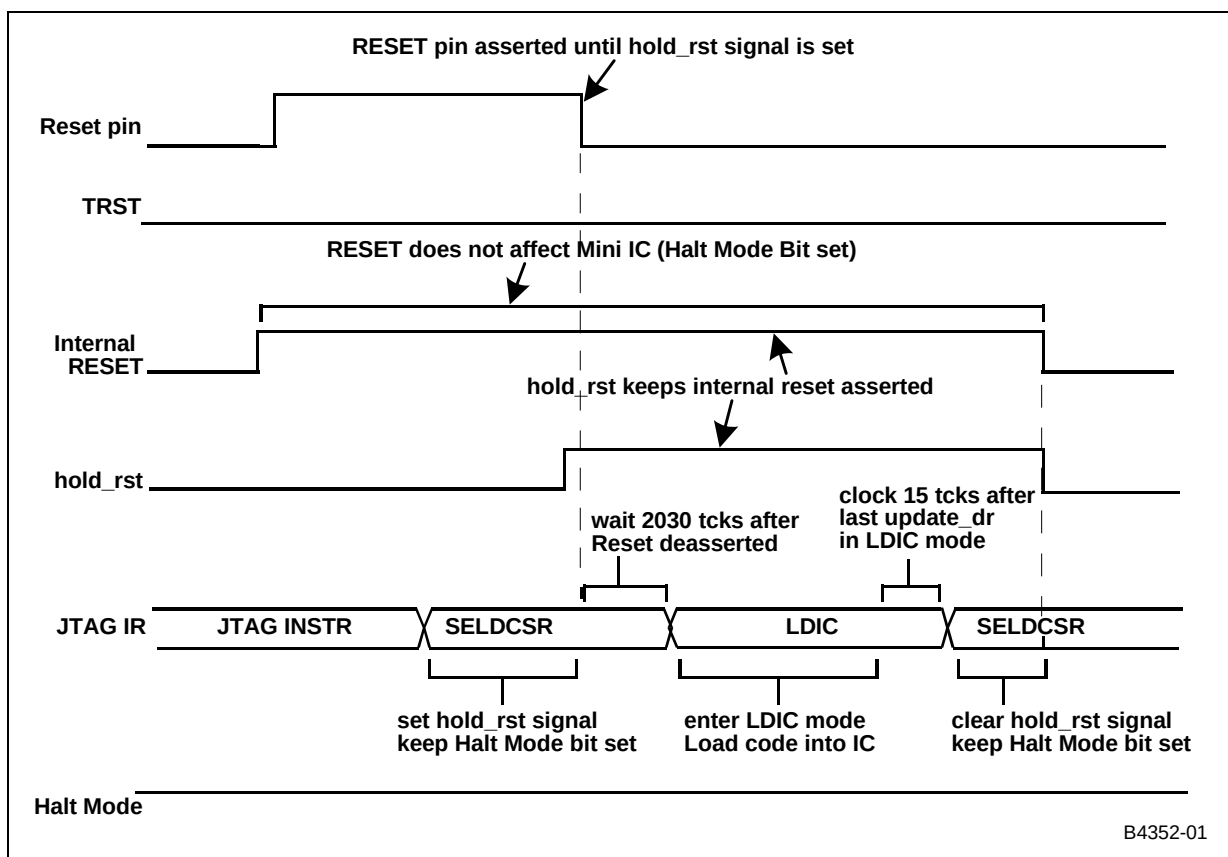
3.6.14.4.2 Loading IC During a Warm Reset for Debug

Loading the instruction cache during a warm reset may be a slightly different situation than during a cold reset. For a warm reset, the main issue is whether the instruction cache gets invalidated by the processor reset or not. There are several possible scenarios:

- While reset is asserted, TRST is also asserted.
In this case the instruction cache is invalidated, so the actions taken to download code are identical to those described in [“Loading IC During Cold Reset for Debug” on page 143](#)
- When reset is asserted, TRST is not asserted, but the processor is not in Halt Mode.
In this case, the instruction cache is also invalidated, so the actions are the same as described in [“Loading IC During Cold Reset for Debug” on page 143](#), after the LDIC instruction is loaded into the JTAG IR.
- When reset is asserted, TRST is not asserted, and the processor is in Halt Mode.
In this last scenario, the mini instruction cache does not get invalidated by reset, since the processor is in Halt Mode. This scenario is described in more detail in this section.

In the last scenario described above is shown in [Figure 26](#).

Figure 25. Code Download During a Warm Reset For Debug



As shown in [Figure 25](#), reset does not invalidate the instruction cache because the processor is in Halt Mode. Since the instruction cache was not invalidated, it may contain valid lines. The host must avoid downloading code to virtual addresses that are already valid in the instruction cache (mini IC or main IC), otherwise the processor may behave unpredictably.

There are several possible solutions that ensure code is not downloaded to a VA that already exists in the instruction cache.

Since the mini instruction cache was not invalidated, any code previously downloaded into the mini IC is valid in the mini IC, so it is not necessary to download the same code again.

If it is necessary to download code into the instruction cache:

1. Assert TRST.
This clears the Halt Mode bit allowing the instruction cache to be invalidated.
2. Clear the Halt Mode bit through JTAG.
This allows the instruction cache to be invalidated by reset.
3. Place the LDIC JTAG instruction in the JTAG IR, then proceed with the normal code download, using the Invalidate IC Line function before loading each line.
This requires 10 packets to be downloaded per cache line instead of the 9 packets described in ["LDIC Cache Functions" on page 141](#)



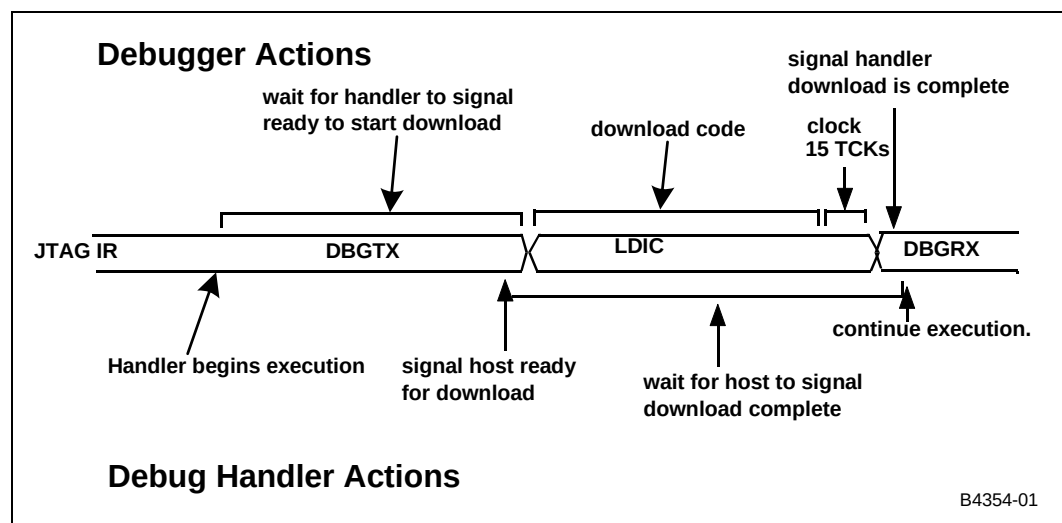
3.6.14.5 Dynamically Loading IC After Reset

An external host can load code into the instruction cache “on the fly” or “dynamically.” This occurs when the host downloads code while the processor is not being reset. However, this requires strict synchronization between the code running on the IXP45X/IXP46X network processors and the external host. The guidelines for downloading code during program execution must be followed to ensure proper operation of the processor. The description in this section focuses on using a debug handler running on the IXP45X/IXP46X network processors to synchronize with the external host, but the details apply for any application that is running while code is dynamically downloaded.

To dynamically download code during software debug, there must be a minimal debug handler stub, responsible for doing the handshaking with the host, resident in the instruction cache. This debug handler stub should be downloaded into the instruction cache during processor reset using the method described in “Loading IC During Reset” on page 142. “Dynamic Code Download Synchronization” on page 148 describes the details for implementing the handshaking in the debug handler.

Figure 26 shows a high level view of the actions taken by the host and debug handler during dynamic code download.

Figure 26. Downloading Code in IC During Program Execution



The following steps describe the details for downloading code:

- Since the debug handler is responsible for synchronization during the code download, the handler must be executing before the host can begin the download. The debug handler execution starts when the application running on the IXP45X/IXP46X network processors generates a debug exception or when the host generates an external debug break.
- While the DBGTX JTAG instruction is in the JTAG IR (see “DBGTX JTAG Command” on page 127), the host polls DBG_SR[0], waiting for the debug handler to set it.
- When the debug handler gets to the point where it is OK to begin the code download, it writes to TX, which automatically sets DBG_SR[0]. This signals the host it is OK to begin the download. The debug handler then begins polling TXRXCTRL[31] waiting for the host to clear it through the DBGRX JTAG register (to indicate the download is complete).
- The host writes LDIC to the JTAG IR, and downloads the code. For each line downloaded, the host must invalidate the target line before downloading code to



that line. Failure to invalidate a line prior to writing it may cause unpredictable operation by the processor.

- When the host completes its download, the host must wait a minimum of 15 TCKs, then switch the JTAG IR to DBGRX, and complete the handshaking (by scanning in a value that sets DBG_SR[35]). This clears TXRXCTL[31] and allows the debug handler code to exit the polling loop. The data scanned into DBG_SR[34:3] is implementation specific.
- After the handler exits the polling loop, it branches to the downloaded code.

Note that this debug handler stub must reside in the instruction cache and execute out of the cache while doing the synchronization. The processor should not be doing any code fetches to external memory while code is being downloaded.

3.6.14.5.1 Dynamic Code Download Synchronization

The following pieces of code are necessary in the debug handler to implement the synchronization used during dynamic code download. The pieces must be ordered in the handler as shown below.



Table 53. Debug-Handler Code to Implement Synchronization During Dynamic Code Download

```
# Before the download can start, all outstanding instruction fetches must complete.
# The MCR invalidate IC by line function serves as a barrier instruction in
# the core. All outstanding instruction fetches are guaranteed to complete before
# the next instruction executes.

# NOTE1: the actual address specified to invalidate is implementation defined, but
# must not have any harmful effects.

# NOTE2: The placement of the invalidate code is implementation defined, the only
# requirement is that it must be placed such that by the time the debugger starts
# loading the instruction cache, all outstanding instruction fetches have completed

    mov r5, address

    mcr p15, 0, r5, c7, c5, 1

# The host waits for the debug handler to signal that it is ready for the
# code download. This can be done using the TX register access handshaking
# protocol. The host polls the TR bit through JTAG until it is set, then begins
# the code download. The following MCR does a write to TX, automatically
# setting the TR bit.

# NOTE: The value written to TX is implementation defined.

    mcr p14, 0, r6, c8, c0, 0

# The debug handler waits until the download is complete before continuing. The
# debugger uses the RX handshaking to signal the debug handler when the download
# is complete. The debug handler polls the RR bit until it is set. A debugger write
# to RX automatically sets the RR bit, allowing the handler to proceed.

# NOTE: The value written to RX by the debugger is implementation defined - it can be a bogus
# value signalling the handler to continue or it can be a target address for the handler to
# branch to.

loop:

    mrc    p14, 0, r15, c14, c0, 0        @ handler waits for signal from debugger
    bpl    loop

    mrc    p14, 0, r0, c8, c0, 0          @ debugger writes target address to RX
    bx     r0
```

In a very simple debug handler stub, the above parts may form the complete handler downloaded during reset (with some handler entry and exit code). When a debug exception occurs, routines can be downloaded as necessary. This basically allows the entire handler to be dynamic.

Another possibility is for a more complete debug handler is downloaded during reset. The debug handler may support some operations, such as read memory, write memory, etc. However, other operations, such as reading or writing a group of CP register, can



be downloaded dynamically. This method could be used to dynamically download infrequently used debug handler functions, while the more common operations remain static in the mini-instruction cache.

The Intel Debug Handler is a complete debug handler that implements the more commonly used functions, and allows less frequently used functions to be dynamically downloaded.

3.6.14.6 Mini-Instruction Cache Overview

The mini instruction cache is a smaller version of the main instruction cache. (For more details on the main instruction cache, see [“Instruction Cache” on page 77.](#)) It is a 2-KByte, two-way set associative cache. There are 32 sets, each containing two ways with each way containing eight words. The cache uses the round-robin replacement policy.

The mini instruction cache is virtually addressed and addresses may be remapped by the PID. However, since the debug handler executes in Special Debug State, address translation and PID remapping are turned off. For application code, accesses to the mini instruction cache use the normal address translation and PID mechanisms.

Normal application code is never cached in the mini instruction cache on an instruction fetch. The only way to get code into the mini instruction cache is through the JTAG LDIC function. Code downloaded into the mini instruction cache is essentially locked - it cannot be overwritten by application code running on the IXP45X/IXP46X network processors. However, it is not locked against code downloaded through the JTAG LDIC functions.

Application code can invalidate a line in the mini instruction cache using a CP15 Invalidate IC line function to an address that hits in the mini instruction cache. However, a CP15 global invalidate IC function does not affect the mini instruction cache.

The mini instruction cache can be globally invalidated through JTAG by the LDIC Invalidate IC function or by a processor reset when the processor is not in HALT or LDIC mode. A single line in the mini instruction cache can be invalidated through JTAG by the LDIC Invalidate IC-line function.

3.6.15 Halt Mode Software Protocol

This section describes the overall debug process in Halt Mode. It describes how to start and end a debug session and details for implementing a debug handler. Intel provides a standard Debug Handler that implements some of the techniques in this section. The Intel Debug Handler itself is a document describing additional handler implementation techniques and requirements.

3.6.15.1 Starting a Debug Session

Prior to starting a debug session in Halt Mode, the debugger must download code into the instruction cache during reset, via JTAG. ([“Downloading Code in ICache” on page 139](#)). This downloaded code should consist of:

- A debug handler
- An override default vector table
- An override relocated vector table (if necessary)

While the processor is still in reset, the debugger should set up the DCSR to trap the reset vector. This causes a debug exception to occur immediately when the processor comes out of reset. Execution is redirected to the debug handler allowing the debugger



to perform any necessary initialization. The reset vector trap is the only debug exception that can occur with debug globally disabled (DCSR[31]=0). Therefore, the debugger must also enable debug prior to existing the handler to ensure all subsequent debug exceptions correctly break to the debug handler.

3.6.15.1.1 Setting up Override Vector Tables

The override default vector table intercepts the reset vector and branches to the debug handler when a debug exception occurs. If the vector table is relocated, the debug vector is relocated to address 0xffff0000. Thus, an override relocated vector table is required to intercept vector 0xffff0000 and branch to the debug handler.

Both override vector tables also intercept the other debug exceptions, so they must be set up to either branch to a debugger specific handler or go to the application's handlers.

It is possible that the application modifies its vector table in memory, so the debugger may not be able to set up the override vector table to branch to the application's handlers. The Debug Handler may be used to work around this problem by reading memory and branching to the appropriate address. Vector traps can be used to get to the debug handler, or the override vector tables can redirect execution to a debug handler routine that examines memory and branches to the application's handler.

3.6.15.1.2 Placing the Handler in Memory

The debug handler is not required to be placed at a specific pre-defined address. However, there are some limitations on where the handler can be placed due to the override vector tables and the two-way set associative mini instruction cache.

In the override vector table, the reset vector must branch to the debug handler using:

- A direct branch, which limits the start of the handler code to within 32 Mbytes of the reset vector, or
- An indirect branch with a data processing instruction. The data processing instruction creates an address using immediate operands and then branches to the target. An LDR to the PC does not work because the debugger cannot set up data in memory before starting the debug handler

The two-way set associative limitation is due to the fact that when the override default and relocated vector tables are downloaded, they take up both ways of Set 0 (w/ addresses 0x0 and 0xffff0000). Therefore, debug handler code cannot be downloaded to an address that maps into Set 0, otherwise it will overwrite one of the vector tables (avoid addresses w/ lower 12 bits=0).

The instruction cache two-way set limitation is not a problem when the reset vector uses a direct branch, since the branch offset can be adjusted accordingly. However, it makes using indirect branches more complicated. Now, the reset vector actually needs multiple data processing instructions to create the target address and branch to it.

One possibility is to set up vector traps on the non-reset exception vectors. These vector locations can then be used to extend the reset vector.

Another solution is to have the reset vector do a direct branch to some intermediate code. This intermediate code can then use several instructions to create the debug handler start address and branch to it. This would require another line in the mini instruction cache, since the intermediate code must also be downloaded. This method also requires that the layout of the debug handler be well thought out to avoid the intermediate code overwriting a line of debug handler code, or vice versa.



For the indirect branch cases, a temporary scratch register may be necessary to hold intermediate values while computing the final target address. DBG_r13 can be used for this purpose (see [“Debug Handler Restrictions” on page 152](#) for restrictions on DBG_r13 usage).

3.6.15.2 Implementing a Debug Handler

The debugger uses the debug handler to examine or modify processor state by sending commands and reading data through JTAG. The API between the debugger and debug handler is specific to a debugger implementation. Intel provides a standard debug handler and API which can be used by third-party vendors. Issues and details for writing a debug handler are discussed in this section and in the Intel Debug Handler.

3.6.15.2.1 Debug Handler Entry

When the debugger requests an external debug break or is waiting for an internal break, it should poll the TR bit through JTAG to determine when the processor has entered Debug Mode. The debug handler entry code must do a write to TX to signal the debugger that the processor has entered Debug Mode. The write to TX sets the TR bit, signalling the host that a debug exception has occurred and the processor has entered Debug Mode. The value of the data written to TX is implementation defined (debug break message, contents of register to save on host, etc.).

3.6.15.2.2 Debug Handler Restrictions

The Debug Handler executes in Debug Mode which is similar to other privileged processor modes, however, there are some differences. Following are restrictions on Debug Handler code and differences between Debug Mode and other privileged modes.

- The processor is in Special Debug State following a debug exception, and thus has special functionality as described in [“Halt Mode” on page 115](#).
- Although address translation and PID remapping are disabled for instruction accesses (as defined in Special Debug State), data accesses use the normal address translation and PID remapping mechanisms.
- Debug Mode does not have a dedicated stack pointer, DBG_r13. Although DBG_r13 exists, it is not a general-purpose register. Its contents are unpredictable and should not be relied upon across any instructions or exceptions. However, DBG_r13 can be used, by data processing (non RRX) and MCR/MRC instructions, as a temporary scratch register.
- The following instructions should not be executed in Debug Mode, they may result in unpredictable behavior:
 - LDM
 - LDR w/ Rd=PC
 - LDR w/ RRX addressing mode
 - SWP
 - LDC
 - STC
- The handler executes in Debug Mode and can be switched to other modes to access banked registers. The handler must not enter User Mode; any User Mode registers that need to be accessed can be accessed in System Mode. Entering User Mode may cause unpredictable behavior.



3.6.15.2.3 Dynamic Debug Handler

On the IXP45X/IXP46X network processors, the debug handler and override vector tables reside in the 2-KByte, mini instruction cache, separate from the main instruction cache. A “static” Debug Handler is downloaded during reset. This is the base handler code, necessary to do common operations such as handler entry/exit, parse commands from the debugger, read/write Intel® StrongARM® registers, read/write memory, etc.

Some functions may require large amounts of code or may not be used very often. As long as there is space in the mini-instruction cache, these functions can be downloaded as part of the static Debug Handler. However, if space is limited, the debug handler also has a dynamic capability that allows a function to be downloaded when it is needed. There are three methods for implementing a dynamic debug handler (using the mini instruction cache, main instruction cache, or external memory). Each method has their limitations and advantages. [“Dynamically Loading IC After Reset” on page 147](#) describes how to dynamically load the mini or main instruction cache.

- Using the Mini IC

The static debug handler can support a command which can have functionality dynamically mapped to it. This dynamic command does not have any specific functionality associated with it until the debugger downloads a function into the mini instruction cache. When the debugger sends the dynamic command to the handler, new functionality can be downloaded, or the previously downloaded functionality can be used.

There are also variations in which the debug handler supports multiple dynamic commands, each mapped to a different dynamic function; or a single dynamic command that can branch to one of several downloaded dynamic functions based on a parameter passed by the debugger.

Debug Handlers that allow code to be dynamically downloaded into the mini instruction cache must be carefully written to avoid inadvertently overwriting a critical piece of debug handler code. Dynamic code is downloaded to the way pointed to by the round-robin pointer. Thus, it is possible for critical debug handler code to be overwritten, if the pointer does not select the expected way.

To avoid this problem, the debug handler should be written to avoid placing critical code in either way of a set that is intended for dynamic code download. This allows code to be downloaded into either way, and the only code that is overwritten is the previously downloaded dynamic function. This method requires that space within the mini instruction cache be allocated for dynamic download, limiting the space available for the static Debug Handler. Also, the space available may not be suitable for a larger dynamic function.

Once downloaded, a dynamic function essentially becomes part of the Debug Handler. Since it is in the mini instruction cache, it does not get overwritten by application code. It remains in the cache until it is replaced by another dynamic function or the lines where it is downloaded are invalidated.

- Using the Main IC.

The steps for downloading dynamic functions into the main instruction cache is similar to downloading into the mini instruction cache. However, using the main instruction cache has its advantages.

Using the main instruction cache eliminates the problem of inadvertently overwriting static Debug Handler code by writing to the wrong way of a set, since the main and mini instruction caches are separate. The debug handler code does not need to be specially mapped out to avoid this problem. Also, space for dynamic functions does not need to be allocated in the mini instruction cache and dynamic functions are not limited to the size allocated.

The dynamic function can actually be downloaded anywhere in the address space. The debugger specifies the location of the dynamic function by writing the address to RX when it signals to the handler to continue. The debug handler then does a branch-and-link to that address.



If the dynamic function is already downloaded in the main instruction cache, the debugger immediately downloads the address, signalling the handler to continue. The static Debug Handler only needs to support one dynamic function command. Multiple dynamic functions can be downloaded to different addresses and the debugger uses the function's address to specify which dynamic function to execute. Since the dynamic function is being downloaded into the main instruction cache, the downloaded code may overwrite valid application code, and conversely, application code may overwrite the dynamic function. The dynamic function is only guaranteed to be in the cache from the time it is downloaded to the time the debug handler returns to the application (or the debugger overwrites it).

- External memory

Dynamic functions can also be downloaded to external memory (or they may already exist there). The debugger can download to external memory using the write-memory commands. Then the debugger executes the dynamic command using the address of the function to identify which function to execute. This method has many of the same advantages as downloading into the main instruction cache.

Depending on the memory system, this method could be much slower than downloading directly into the instruction cache. Another problem is the application may write to the memory where the function is downloaded. If it can be guaranteed that the application does not modify the downloaded dynamic function, the debug handler can save the time it takes to re-download the code. Otherwise, to ensure the application does not corrupt the dynamic functions, the debugger should re-download any dynamic functions it uses.

For all three methods, the downloaded code executes in the context of the debug handler. The processor will be in Special Debug State, so all of the special functionality applies.

The downloaded functions may also require some common routines from the static debug handler, such as the polling routines for reading RX or writing TX. To simplify the dynamic functions, the debug handler should define a set of registers to contain the addresses of the most commonly used routines. The dynamic functions can then access these routines using indirect branches (BLX). This helps reduce the amount of code in the dynamic function since common routines do not need to be replicated within each dynamic function.

3.6.15.2.4 High-Speed Download

Special debug hardware has been added to support a high-speed download mode to increase the performance of downloads to system memory (vs. writing a block of memory using the standard handshaking).

The basic assumption is that the debug handler can read any data sent by the debugger and write it to memory, before the debugger can send the next data. Thus, in the time it takes for the debugger to scan in the next data word and do an Update_DR, the handler is already in its polling loop, waiting for it. Using this assumption, the debugger does not have to poll RR to see whether the handler has read the previous data - it assumes the previous data has been consumed and immediately starts scanning in the next data word.

The pitfall is when the write to memory stalls long enough that the assumption fails. In this case the download with normal handshaking can be used (or high-speed download can still be used, but a few extra TCKs in the Pause_DR state may be necessary to allow a little more time for the store to complete).

The hardware support for high-speed download includes the Download bit (DCSR[29]) and the Overflow Flag (DCSR[30]).



The download bit acts as a branch flag, signalling to the handler to continue with the download. This removes the need for a counter in the debug handler.

The overflow flag indicates that the debugger attempted to download the next word before the debugger read the previous word.

More details on the Download bit, Overflow flag and high-speed download, in general, can be found in [“Transmit/Receive Control Register” on page 120](#).

Following is example code showing how the Download bit and Overflow flag are used in the debug handler:

Table 54. Debug Handler Code: Download Bit and Overflow Flag

```

hs_write_word_loop:
hs_write_overflow:
    bl    read_RX            @ read data word from host

    @@ read TXRXCTRL into the CCs
    mrc    p14, 0, r15, c14, c0, 0
    bcc    hs_write_done     @ if D bit clear, download complete, exit loop.
    beq    hs_write_overflow @ if overflow detected, loop until host clears D bit

    str    r0, [r6], #4      @ store only if there is no overflow.

    b      hs_write_word_loop @ get next data word

hs_write_done:
    @@ after the loop, if the overflow flag was set, return error message to host
    moveq   r0, #OVERFLOW_RESPONSE
    beq     send_response
    b       write_common_exit

```

3.6.15.3 Ending a Debug Session

Prior to ending a debug session, the debugger should take the following actions:

1. Clear the DCSR (disable debug, exit Halt Mode, clear all vector traps, disable the trace buffer)
2. Turn off all breakpoints.
3. Invalidate the mini instruction cache.
4. Invalidate the main instruction cache.
5. Invalidate the BTB.



These actions ensure that the application program executes correctly after the debugger has been disconnected.

3.6.16 Software Debug Notes and Errata

- Trace buffer message count value on data aborts:
LDR to non-PC that aborts gets counted in the exception message. But an LDR to the PC that aborts does not get counted on exception message.
- SW Note on data abort generation in Special Debug State.
 - Avoid code that could generate precise data aborts.
 - If this cannot be done, then handler needs to be written such that a memory access is followed by 1 nops. In this case, certain memory operations must be avoided - LDM, STM, STRD, LDC, SWP.
- Data abort on Special Debug State:
When write-back is on for a memory access that causes a data abort, the base register is updated with the write-back value. This is inconsistent with normal (non-SDS) behavior where the base remains unchanged if write-back is on and a data abort occurs.
- Trace Buffer wraps around and loses data in Halt Mode when configured for fill-once mode:
It is possible to overflow (and lose) data from the trace buffer in fill-once mode, in Halt Mode. When the trace buffer fills up, it has space for 1 indirect branch message (5 bytes) and 1 exception message (1 Byte).
If the trace buffer fills up with an indirect branch message and generates a trace buffer full break at the same time as a data abort occurs, the data abort has higher priority, so the processor first goes to the data abort handler. This data abort is placed into the trace buffer without losing any data.
However, if another imprecise data abort is detected at the start of the data abort handler, it will have higher priority than the trace buffer full break, so the processor will go back to the data abort handler. This 2nd data abort also gets written into the trace buffer. This causes the trace buffer to wrap-around and one trace buffer entry is lost (oldest entry is lost). Additional trace buffer entries can be lost if imprecise data aborts continue to be detected before the processor can handle the trace buffer full break (which will turn off the trace buffer).
This trace buffer overflow problem can be avoided by enabling vector traps on data aborts.
- TXRXCTRL.RR prevents TX register from being updated (even if TXRXCTRL.TR is clear). This will be fixed on B-step.
The problem is that there is incorrect (and unnecessary) interaction between the RX ready (RR) flag and writing the TX register. The debug handler looks at the TX ready bit before writing to the TX register. If this bit is clear, then the handler should be able to write to the TX register. However, in the current implementation even if the TR bit is clear, if the RR bit is set, TX will be unchanged when the handler writes to it. It is OK to prevent a write to TX when the TR bit is set (since the host has not read the previous data in the TX, and we don't want a write to TX to overwrite previous data).
- The TXRXCTRL.OV bit (overflow flag) does not get set during high-speed download when the handler reads the RX register at the same time the debugger writes to it. If the debugger writes to RX at the same time the handler reads from RX, the handler read returns the newly written data and the previous data is lost. However, in this specific case, the overflow flag does not get set, so the debugger is unaware that the download was not successful.



3.7 Performance Monitoring

This section describes the performance monitoring facility of the IXP45X/IXP46X network processors. The events that are monitored can provide performance information for compiler writers, system application developers and software programmers.

3.7.1 Overview

The hardware for the IXP45X/IXP46X network processors provides four 32-bit performance counters that allow four unique events to be monitored simultaneously. In addition, the IXP45X/IXP46X network processors implement a 32-bit clock counter that can be used in conjunction with the performance counters; its main purpose is to count the number of core clock cycles which is useful in measuring total execution time.

The IXP45X/IXP46X network processors can monitor either occurrence events or duration events. When counting occurrence events, a counter is incremented each time a specified event takes place and when measuring duration, a counter counts the number of processor clocks that occur while a specified condition is true. If any of the five counters overflow, an interrupt request will occur if it is enabled. Each counter has its own interrupt request enable. The counters continue to monitor events even after an overflow occurs, until disabled by software.

Each of these counters can be programmed to monitor any one of various events.

To further augment performance monitoring, the IXP45X/IXP46X network processors' clock counter can be used to measure the executing time of an application. This information combined with a duration event can feedback a percentage of time the event occurred with respect to overall execution time.

All of the performance monitoring registers are accessible through Coprocessor 14 (CP14). Refer to [Table 29](#) for more details on accessing these registers with **MRC** and **MCR** coprocessor instructions. Access is allowed in privileged mode only. Note that these registers can't be access with **LDC** or **STC** coprocessor instructions.

Table 55. Performance Monitoring Registers

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0



3.7.2 Register Description

Table 56. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

3.7.2.1 Clock Counter (CCNT)

The format of CCNT is shown in [Table 57](#). The clock counter is reset to '0' by setting bit 2 in the Performance Monitor Control Register (PMNC) or can be set to a predetermined value by directly writing to it. It counts core clock cycles. When CCNT reaches its maximum value 0xFFFF,FFFF, the next clock cycle will cause it to roll over to zero and set the overflow flag (bit 0) in FLAG. An interrupt request will occur if it is enabled via bit 0 in INTEN.

Table 57. Clock Count Register (CCNT)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0																													
Clock Counter																													
reset value: unpredictable																													
Bits	Access		Description																										
31:0	Read / Write		32-bit clock counter - Reset to '0' by PMNC register. When the clock counter reaches its maximum value 0xFFFF,FFFF, the next cycle will cause it to roll over to zero and generate an interrupt request if enabled.																										

3.7.2.2 Performance Count Registers

There are four 32-bit event counters; their format is shown in [Table 58](#). The event counters are reset to '0' by setting bit 1 in the PMNC register or can be set to a predetermined value by directly writing to them. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count will cause it to roll over to zero and set its corresponding overflow flag (bit 1,2,3 or 4) in FLAG. An interrupt request will be generated if its corresponding interrupt enable (bit 1,2,3 or 4) is set in INTEN.

Table 58. Performance Monitor Count Register (PMN0 - PMN3)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0																													
Event Counter																													
reset value: unpredictable																													
Bits	Access		Description																										
31:0	Read / Write		32-bit event counter - Reset to '0' by PMNC register. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count will cause it to roll over to zero and generate an interrupt request if enabled.																										



3.7.2.3 Performance Monitor Control Register

The performance monitor control register (PMNC) is a coprocessor register that:

- Contains the PMU ID
- Extends CCNT counting by six more bits (cycles between counter rollover = 2^{38})
- Resets all counters to zero
- And enables the entire mechanism

Table 59 shows the format of the PMNC register.

Table 59. Performance Monitor Control Register

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

ID		D	C	P	E

reset value: E and ID are 0, others unpredictable

Bits	Access	Description
31:24	Read / Write Ignored	Performance Monitor Identification (ID) - IXP45X/IXP46X network processors = 0x14
23:4	Read-unpredictable / Write-as-0	Reserved
3	Read / Write	Clock Counter Divider (D) - 0 = CCNT counts every processor clock cycle 1 = CCNT counts every 64 th processor clock cycle
2	Read-unpredictable / Write	Clock Counter Reset (C) - 0 = no action 1 = reset the clock counter to '0x0'
1	Read-unpredictable / Write	Performance Counter Reset (P) - 0 = no action 1 = reset all performance counters to '0x0'
0	Read / Write	Enable (E) - 0 = all counters are disabled 1 = all counters are enabled

3.7.2.4 Interrupt Enable Register

Each counter can generate an interrupt request when it overflows. INTEN enables interrupt requesting for each counter.

Table 60. Interrupt Enable Register (Sheet 1 of 2)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

reset value: [4:0] = 0b00000, others unpredictable

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	PMN3 Interrupt Enable (P3) - 0 = disable interrupt 1 = enable interrupt
3	Read / Write	PMN2 Interrupt Enable (P2) - 0 = disable interrupt 1 = enable interrupt

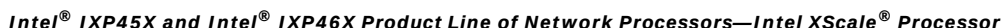


Table 60. Interrupt Enable Register (Sheet 2 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											P ₃	P ₂	P ₁	P ₀	C

reset value: [4:0] = 0b00000, others unpredictable

Bits	Access	Description
2	Read / Write	PMN1 Interrupt Enable (P1) - 0 = disable interrupt 1 = enable interrupt
1	Read / Write	PMN0 Interrupt Enable (P0) - 0 = disable interrupt 1 = enable interrupt
0	Read / Write	CCNT Interrupt Enable (C) - 0 = disable interrupt 1 = enable interrupt

3.7.2.5 Overflow Flag Status Register

FLAG identifies which counter has overflowed and also indicates an interrupt has been requested if the overflowing counter's corresponding interrupt enable bit (contained within INTEN) is asserted. An overflow is cleared by writing a '1' to the overflow bit.

Table 61. Overflow Flag Status Register (Sheet 1 of 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											P ₃	P ₂	P ₁	P ₀	C

reset value: [4:0] = 0b00000, others unpredictable

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	PMN3 Overflow Flag (P3) - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
3	Read / Write	PMN2 Overflow Flag (P2) - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit

**Table 61. Overflow Flag Status Register (Sheet 2 of 2)**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																											P3	P2	P1	P0	C

reset value: [4:0] = 0b000000, others unpredictable

Bits	Access	Description
2	Read / Write	PMN1 Overflow Flag (P1) - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
1	Read / Write	PMN0 Overflow Flag (P0) - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
0	Read / Write	CCNT Overflow Flag (C) - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit

3.7.2.6 Event Select Register

EVTSEL is used to select events for PMN0, PMN1, PMN2 and PMN3. Refer to [Table 63](#), “Performance Monitoring Events” on page 162 for a list of possible events.

Table 62. Event Select Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
evtCount3								evtCount2								evtCount1								evtCount0							

reset value: unpredictable

Bits	Access	Description
31:24	Read / Write	Event Count 3 (evtCount3) - Identifies the source of events that PMN3 counts. See Table 63 for a description of the values this field may contain.
23:16	Read / Write	Event Count 2 (evtCount2) - Identifies the source of events that PMN2 counts. See Table 63 for a description of the values this field may contain.
15:8	Read / Write	Event Count 1 (evtCount1) - Identifies the source of events that PMN1 counts. See Table 63 for a description of the values this field may contain.
7:0	Read / Write	Event Count 0 (evtCount0) - Identifies the source of events that PMN0 counts. See Table 63 for a description of the values this field may contain.



3.7.3 Managing the Performance Monitor

The following are a few notes about controlling the performance monitoring mechanism:

- An interrupt request will be generated when a counter's overflow flag is set and its associated interrupt enable bit is set in INTEN. The interrupt request will remain asserted until software clears the overflow flag by writing a one to the flag that is set. (Note that the product specific interrupt unit and the CPSR must have enabled the interrupt in order for software to receive it.) The interrupt request can also be deasserted by clearing the corresponding interrupt enable bit. Disabling the facility (PMNC.E) doesn't deassert the interrupt request.
- The counters continue to record events even after they overflow.
- To change an event for a performance counter, first disable the facility (PMNC.E) and then modify EVTSEL. Not doing so will cause unpredictable results.
- To increase the monitoring duration, software can extend the count duration beyond 32 bits by counting the number of overflow interrupts each 32-bit counter generates. This can be done in the interrupt service routine (ISR) where an increment to some memory location every time the interrupt occurs will enable longer durations of performance monitoring. This does intrude upon program execution but is negligible, since the ISR execution time is in the order of tens of cycles compared to the number of cycles it took to generate an overflow interrupt (2^{32}).
- Power can be saved by selecting event 0xFF for any unused event counter. This only applies when other event counters are in use. When the performance monitor is not used at all (PMNC.E = 0x0), hardware ensures minimal power consumption.

3.7.4 Performance Monitoring Events

Table 63 lists events that may be monitored. Each of the Performance Monitor Count Registers (PMN0, PMN1, PMN2, and PMN3) can count any listed event. Software selects which event is counted by each PMNx register by programming the evtCountx fields of EVTSEL.

Table 63. Performance Monitoring Events (Sheet 1 of 2)

Event Number (evtCountx)	Event Definition
0x0	Instruction cache miss requires fetch from external memory.
0x1	Instruction cache cannot deliver an instruction. This could indicate an ICache miss or an ITLB miss. This event will occur every cycle in which the condition is present.
0x2	Stall due to a data dependency. This event will occur every cycle in which the condition is present.
0x3	Instruction TLB miss.
0x4	Data TLB miss.
0x5	Branch instruction executed, branch may or may not have changed program flow. (Counts only B and BL instructions, in both Intel® StrongARM® and Thumb mode.)
0x6	Branch incorrectly predicted. (Counts only B and BL instructions, in both Intel® StrongARM® and Thumb mode.)
0x7	Instruction executed.
0x8	Stall because the data cache buffers are full. This event will occur every cycle in which the condition is present.
0x9	Stall because the data cache buffers are full. This event will occur once for each contiguous sequence of this type of stall.
0xA	Data cache access, not including Cache Operations (defined in "Register 7: Cache Functions" on page 103)

**Table 63. Performance Monitoring Events (Sheet 2 of 2)**

Event Number (evtCountx)	Event Definition
0xB	Data cache miss, not including Cache Operations (defined in "Register 7: Cache Functions" on page 103)
0xC	Data cache write-back. This event occurs once for each 1/2 line (four words) that are written back from the cache.
0xD	Software changed the PC. All 'b', 'bl', 'blx', 'mov[s] pc, Rm', 'ldm Rn, {Rx, pc}', 'ldr pc, [Rm]', 'pop {pc}' will be counted. An 'mcr p<cp>, 0, pc, ...', will not. The count also does not increment when an event occurs and the PC changes to the event address, e.g., IRQ, FIQ, SWI, etc.
0x10 through 0x17	Reserved.
all others	Reserved, unpredictable results

Some typical combinations of counted events are listed in this section and summarized in Table 64. In this section, we call such an event combination a *mode*.

Table 64. Common Uses of the PMU

Mode	EVTSEL.evtCount0	EVTSEL.evtCount1
Instruction Cache Efficiency	0x7 (instruction count)	0x0 (ICache miss)
Data Cache Efficiency	0xA (Dcache access)	0xB (DCache miss)
Instruction Fetch Latency	0x1 (ICache cannot deliver)	0x0 (ICache miss)
Data/Bus Request Buffer Full	0x8 (DBuffer stall duration)	0x9 (DBuffer stall)
Stall/Writeback Statistics	0x2 (data stall)	0xC (DCache writeback)
Instruction TLB Efficiency	0x7 (instruction count)	0x3 (ITLB miss)
Data TLB Efficiency	0xA (Dcache access)	0x4 (DTLB miss)

Note: PMN0 and PMN1 were used for illustration purposes only. Given there are four event counters, more elaborate combination of events could be constructed. For example, one performance run could select 0xA, 0xB, 0xC, 0x9 events from which data cache performance statistics could be gathered (like hit rates, number of write-backs per data cache miss, and number of times the data cache buffers fill up per request).

3.7.4.1 Instruction Cache Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions fetched from the instruction cache that were never executed. This can happen if a branch instruction changes the program flow; the instruction cache may retrieve the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time.

Statistics derived from these two events:

- Instruction cache miss-rate. This is derived by dividing PMN1 by PMN0.
- *The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI).* CPI can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

3.7.4.2 Data Cache Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.



Note that **STM** and **LDM** will each count as several accesses to the data cache depending on the number of registers specified in the register list. **LDRD** will register two accesses.

PMN1 counts the number of data cache and mini-data cache misses. Cache operations do not contribute to this count. See “[Register 7: Cache Functions](#)” on page 103 for a description of these operations.

The statistic derived from these two events is:

Data cache miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.4.3 Instruction Fetch Latency Mode

PMN0 accumulates the number of cycles when the instruction-cache is not able to deliver an instruction to the IXP45X/IXP46X network processors due to an instruction-cache miss or instruction-TLB miss. This event means that the processor core is stalled.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time. This is the same event as measured in instruction cache efficiency mode.

Statistics derived from these two events:

- *The average number of cycles the processor stalled waiting for an instruction fetch from external memory to return.* This is calculated by dividing PMN0 by PMN1. If the average is high, then the IXP45X/IXP46X network processors may be starved of the bus external to the IXP45X/IXP46X network processors.
- *The percentage of total execution cycles the processor stalled waiting on an instruction fetch from external memory to return.* This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time.

3.7.4.4 Data/Bus Request Buffer Full Mode

The Data Cache has buffers available to service cache misses or uncacheable accesses. For every memory request that the Data Cache receives from the processor core a buffer is speculatively allocated in case an external memory request is required or temporary storage is needed for an unaligned access. If no buffers are available, the Data Cache will stall the processor core. How often the Data Cache stalls depends on the performance of the bus external to the IXP45X/IXP46X network processors and what the memory access latency is for Data Cache miss requests to external memory. If the memory access latency for the IXP45X/IXP46X network processors is high, possibly due to starvation, these Data Cache buffers will become full. This performance monitoring mode is provided to see if the IXP45X/IXP46X network processors are being starved of the bus external to the IXP45X/IXP46X network processors, which will affect the performance of the application running on the IXP45X/IXP46X network processors.

PMN0 accumulates the number of clock cycles the processor is being stalled due to this condition and PMN1 monitors the number of times this condition occurs.

Statistics derived from these two events:

- *The average number of cycles the processor stalled on a data-cache access that may overflow the data-cache buffers.* This is calculated by dividing PMN0 by PMN1. This statistic lets you know if the duration event cycles are due to many requests or are attributed to just a few requests. If the average is high, the IXP45X/IXP46X network processors may be starved of the bus external to the IXP45X/IXP46X network processors.
- *The percentage of total execution cycles the processor stalled because a Data Cache request buffer was not available.* This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time.



3.7.4.5 Stall / Write-Back Statistics

When an instruction requires the result of a previous instruction and that result is not yet available, the IXP45X/IXP46X network processors stall in order to preserve the correct data dependencies. PMN0 counts the number of stall cycles due to data-dependencies. Not all data-dependencies cause a stall; only the following dependencies cause such a stall penalty:

- Load-use penalty: attempting to use the result of a load before the load completes. To avoid the penalty, software should delay using the result of a load until it is available. This penalty shows the latency effect of data-cache access.
- Multiply/Accumulate-use penalty: attempting to use the result of a multiply or multiply-accumulate operation before the operation completes. Again, to avoid the penalty, software should delay using the result until it is available.
- ALU use penalty: there are a few isolated cases where back-to-back ALU operations may result in one cycle delay in the execution. These cases are defined in [Table 3.9, “Performance Considerations” on page 181.](#)

PMN1 counts the number of write-back operations emitted by the data cache. These write-backs occur when the data cache evicts a dirty line of data to make room for a newly requested line or as the result of clean operation (CP15, register 7).

Statistics derived from these two events:

- *The percentage of total execution cycles the processor stalled because of a data dependency.* This is calculated by dividing PMN0 by CCNT, which was used to measure total execution time. Often a compiler can reschedule code to avoid these penalties when given the right optimization switches.
- Total number of data write-back requests to external memory can be derived solely with PMN1.

3.7.4.6 Instruction TLB Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions that were translated by the instruction TLB and never executed. This can happen if a branch instruction changes the program flow; the instruction TLB may translate the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction TLB table-walks, which occurs when there is a TLB miss. If the instruction TLB is disabled PMN1 will not increment.

Statistics derived from these two events:

- Instruction TLB miss-rate. This is derived by dividing PMN1 by PMN0.
- *The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI).* CPI can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

3.7.4.7 Data TLB Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note that **STM** and **LDM** will each count as several accesses to the data TLB depending on the number of registers specified in the register list. **LDRD** will register two accesses.



PMN1 counts the number of data TLB table-walks, which occurs when there is a TLB miss. If the data TLB is disabled PMN1 will not increment.

The statistic derived from these two events is:

Data TLB miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.5 Multiple Performance Monitoring Run Statistics

There may be times when more than four events need to be monitored for performance tuning. In this case, multiple performance monitoring runs can be done, capturing different events from each run. For example, the first run could monitor the events associated with instruction cache performance and the second run could monitor the events associated with data cache performance. By combining the results, statistics like total number of memory requests could be derived.

3.7.6 Examples

In this example, the events selected with the Instruction Cache Efficiency mode are monitored and CCNT is used to measure total execution time. Sampling time ends when PMN0 overflows which will generate an IRQ interrupt.

Example 14. Configuring the Performance Monitor

```
; Configure the performance monitor with the following values:
; EVTSEL.evtCount0 = 7, EVTSEL.evtCount1 = 0 instruction cache efficiency
; INTEN.inten = 0x7 set all counters to trigger an interrupt on overflow
; PMNC.C = 1 reset CCNT register
; PMNC.P = 1 reset PMN0 and PMN1 registers
; PMNC.E = 1 enable counting
MOV R0,#0x700
MCR P14,0,R0,C8,C1,0 ; setup EVTSEL
MOV R0,#0x7
MCR P14,0,R0,C4,C1,0 ; setup INTEN
MCR P14,0,R0,C0,C1,0 ; setup PMNC, reset counters & enable
; Counting begins
```

Counter overflow can be dealt with in the IRQ interrupt service routine as shown below:



Example 15. Interrupt Handling

```

IRQ_INTERRUPT_SERVICE_ROUTINE:

; Assume that performance counting interrupts are the only IRQ in the system

MRC P14,0,R1,C0,c1,0 ; read the PMNC register
BIC R2,R1,#1          ; clear the enable bit, preserve other bits in PMNC
MCR P14,0,R2,C0,c1,0 ; disable counting
MRC P14,0,R3,C1,c1,0 ; read CCNT register
MRC P14,0,R4,C0,c2,0 ; read PMN0 register
MRC P14,0,R5,C1,c2,0 ; read PMN1 register

<process the results>

SUBS PC,R14,#4        ; return from interrupt

```

As an example, assume the following values in CCNT, PMN0, PMN1 and PMNC:

Example 16. Computing the Results

```

; Assume CCNT overflowed

CCNT = 0x0000,0020 ;Overflowed and continued counting
Number of instructions executed = PMN0 = 0x6AAA,AAAA
Number of instruction cache miss requests = PMN1 = 0x0555,5555
Instruction Cache miss-rate = 100 * PMN1/PMN0 = 5%
CPI = (CCNT + 2^32)/Number of instructions executed = 2.4 cycles/instruction

```

In the contrived example above, the instruction cache had a miss-rate of 5% and CPI was 2.4.

3.8 Programming Model

This section describes the programming model of the IXP45X/IXP46X network processors, namely the implementation options and extensions to the Intel® StrongARM® V5TE architecture.

3.8.1 Intel® StrongARM® V5TE Architecture Compatibility

The IXP45X/IXP46X network processors implement the integer instruction set architecture specified in Intel® StrongARM® V5TE. T refers to the thumb instruction set and E refers to the DSP-Enhanced instruction set.

Intel StrongARM V5TE introduces a few more architecture features over Intel StrongARM V4, specifically the addition of tiny pages (1 Kbyte), a new instruction that counts the leading zeroes (**CLZ**) in a data value, enhanced Intel StrongARM-Thumb transfer instructions and a modification of the system control coprocessor, CP15.



3.8.2 Intel® StrongARM® Architecture Implementation Options

3.8.2.1 Big-Endian versus Little-Endian

The IXP45X/IXP46X network processors can operate in big or little endian mode. The B-bit of the Control Register, coprocessor 15, register 1, bit 7 (see [Section 3.5.1.2](#)) contained within the IXP45X/IXP46X network processors selects the endianness mode of the Intel XScale processor.

Note: This bit takes effect even if the MMU is disabled.

If you choose little endian then you have further options that control whether address and/or data coherency modes. Refer P-Attribute bit in the MMU (see [Section 3.1.1.1](#), “Page (P) Attribute Bit” on page 70) and Expansion Bus Configuration Register 1, Bit 8, BYTE_SWAP_ENABLE (Table 231, “Expansion Bus Configuration Register 1-Bit Definition” on page 710).

Note: The NPEs on the IXP45X/IXP46X network processors are **Big-Endian only**; so if you change the endianness of the Intel XScale processor to little endian for your operating system, then this has an impact on how the NPEs and Intel XScale processor exchange data. The Intel® IXP400 Software Release handles this.

3.8.2.2 26-Bit Architecture

The Intel XScale processor does not support 26-bit architecture.

3.8.2.3 Thumb

The Intel XScale processor supports the thumb instruction set.

3.8.2.4 Intel® StrongARM® DSP-Enhanced Instruction Set

The Intel XScale processor implements Intel StrongARM's DSP-enhanced instruction set which is a set of instructions that boost the performance of signal processing applications. There are new multiply instructions that operate on 16-bit data values and new saturation instructions. Saturated instructions are used to ensure accuracy during DSP operations to ensure the signed extension is maintained during an overflow arithmetic operation. Further information on saturated integer arithmetic can be found in the *ARM® Architecture Reference Manual*.

Some of the new instructions are:

- SMLAxy — $32 \leq 16 \times 16 + 32$
- SMLAWy — $32 \leq 32 \times 16 + 32$
- SMLALxy — $64 \leq 16 \times 16 + 64$
- SMULxy — $32 \leq 16 \times 16$
- SMULWy — $32 \leq 32 \times 16$
- QADD — Adds two registers and saturates the result if an overflow occurred
- QDADD — Doubles and saturates one of the input registers then add and saturate
- QSUB — Subtracts two registers and saturates the result if an overflow occurred
- QDSUB — Doubles and saturates one of the input registers then subtract and saturate

The Intel XScale processor also implements Load Two words (LDRD), Store Two Words (STRD) and cache preload (PLD) instructions with the following implementation notes:



- PLD is interpreted as a read operation by the MMU and is ignored by the data breakpoint unit, i.e., PLD will never generate data breakpoint events.
- PLD to a non-cacheable page performs no action. Also, if the targeted cache line is already resident, this instruction has no affect.
- Both LDRD and STRD instructions will generate an alignment exception when the address bits [2:0] = 0b100.

The transfers of two Intel StrongARM register values to a coprocessor (MCRR) and the transfer of values from a coprocessor to two Intel StrongARM registers (MRRC) are only supported on the IXP45X/IXP46X network processors when directed to coprocessor 0 and are used to access the internal accumulator. See [“Internal Accumulator Access Format” on page 173](#) for more information. Access to coprocessors 15 and 14 generate an undefined instruction exception.

3.8.2.5 Base Register Update

If a data abort is signalled on a memory instruction that specifies write-back, the contents of the base register will not be updated. This holds for all load and store instructions. This behavior matches that of the first generation Intel StrongARM processor and is referred to in the Intel StrongARM V5TE architecture as the *Base Restored Abort Model*.

3.8.3 Extensions to Intel® StrongARM* Architecture

The Intel XScale processor adds a few extensions to the Intel StrongARM Version 5TE architecture to meet the needs of various markets and design requirements. The following is a list of the extensions which are discussed in the next sections.

- A DSP coprocessor (CP0) has been added that contains a 40-bit accumulator and eight new instructions.
- New page attributes were added to the page table descriptors. The C and B page attribute encoding was extended by one more bit to allow for more encodings: write allocate and mini-data cache.
- Additional functionality has been added to coprocessor 15. Coprocessor 14 was also created.
- Enhancements were made to the Event Architecture, which include instruction cache and data cache parity error exceptions, breakpoint events, and imprecise external data aborts.

3.8.3.1 DSP Coprocessor 0 (CP0)

The Intel XScale processor adds a DSP coprocessor to the architecture for the purpose of increasing the performance and the precision of audio processing algorithms. This coprocessor contains a 40-bit accumulator and eight new instructions.

The 40-bit accumulator is referenced by several new instructions that were added to the architecture; **MIA**, **MIAPH** and **MIAxy** are multiply/accumulate instructions that reference the 40-bit accumulator instead of a register specified accumulator. **MAR** and **MRA** provide the ability to read and write the 40-bit accumulator.

Access to CP0 is always allowed in all processor modes when bit 0 of the Coprocessor Access Register is set. Any access to CP0 when this bit is clear will cause an undefined exception. (See [“Register 15: Coprocessor Access Register” on page 107](#) for more details). Note that only privileged software can set this bit in the Coprocessor Access Register located in CP15.

The 40-bit accumulator will need to be saved on a context switch if multiple processes are using it.



Two new instruction formats were added for coprocessor 0: Multiply with Internal Accumulate Format and Internal Accumulate Access Format. The formats and instructions are described next.

3.8.3.1.1 Multiply with Internal Accumulate Format

A new multiply format has been created to define operations on 40-bit accumulators. Table 9, “MRC/MCR Format” on page 97 shows the layout of the new format. The op code for this format lies within the coprocessor register transfer instruction type. These instructions have their own syntax.

Table 65. Multiply with Internal Accumulate Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	1	0	0	0	1	0	opcode_3				Rs			0	0	0	0	acc				1	Rm					

Bits	Description	Notes
31:28	cond - Intel® StrongARM® condition codes	-
19:16	opcode_3 - specifies the type of multiply with internal accumulate	Intel XScale processor defines the following: 0b0000 = MIA 0b1000 = MIAPH 0b1100 = MIABB 0b1101 = MIABT 0b1110 = MIATB 0b1111 = MIATT The effect of all other encodings are unpredictable.
15:12	Rs - Multiplier	
7:5	acc - select 1 of 8 accumulators	Intel XScale processor only implements acc0; access to any other acc has unpredictable effect.
3:0	Rm - Multiplicand	-

Two new fields were created for this format, *acc* and *opcode_3*. The *acc* field specifies one of eight internal accumulators to operate on and *opcode_3* defines the operation for this format. The Intel XScale processor defines a single 40-bit accumulator referred to as *acc0*; future implementations may define multiple internal accumulators. The Intel XScale processor uses *opcode_3* to define six instructions, **MIA**, **MIAPH**, **MIABB**, **MIABT**, **MIATB** and **MIATT**.

**Table 66. MIA{<cond>} acc0, Rm, Rs**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	Rs	0	0	0	0	0	0	0	0	0	0	1	Rm			

Operation: if ConditionPassed(<cond>) then

$$\text{acc0} = (\text{Rm}[31:0] * \text{Rs}[31:0])[39:0] + \text{acc0}[39:0]$$

Exceptions: none

Qualifiers Condition Code

No condition code flags are updated

Notes: Early termination is supported. Instruction timings can be found in [“Multiply Instruction Timings” on page 185](#).

Specifying R15 for register Rs or Rm has unpredictable results.

acc0 is defined to be 0b000 on Intel XScale processor.

The **MIA** instruction operates similarly to **MLA** except that the 40-bit accumulator is used. **MIA** multiplies the signed value in register Rs (multiplier) by the signed value in register Rm (multiplicand) and then adds the result to the 40-bit accumulator (acc0).

MIA does not support unsigned multiplication; all values in Rs and Rm will be interpreted as signed data values. **MIA** is useful for operating on signed 16-bit data that was loaded into a general-purpose register by **LDRSH**.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 67. MIAPH{<cond>} acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	Rs	0	0	0	0	0	0	0	0	0	0	1	Rm			

Operation: if ConditionPassed(<cond>) then

$$\begin{aligned} \text{acc0} = & \text{sign\_extend}(\text{Rm}[31:16] * \text{Rs}[31:16]) + \\ & \text{sign\_extend}(\text{Rm}[15:0] * \text{Rs}[15:0]) + \\ & \text{acc0}[39:0] \end{aligned}$$

Exceptions: none

Qualifiers Condition Code

S bit is always cleared; no condition code flags are updated

Notes: Instruction timings can be found in [“Multiply Instruction Timings” on page 185](#).

Specifying R15 for register Rs or Rm has unpredictable results.

acc0 is defined to be 0b000 on Intel XScale processor



The **MIAPH** instruction performs two 16-bit signed multiplies on packed half word data and accumulates these to a single 40-bit accumulator. The first signed multiplication is performed on the lower 16 bits of the value in register Rs with the lower 16 bits of the value in register Rm. The second signed multiplication is performed on the upper 16 bits of the value in register Rs with the upper 16 bits of the value in register Rm. Both signed 32-bit products are sign extended and then added to the value in the 40-bit accumulator (acc0).

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 68. MIAxy{<cond>} acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	1	1	x	y	Rs				0	0	0	0	0	0	0	1	Rm			

Operation: if ConditionPassed(<cond>) then

if (bit[17] == 0)

<operand1> = Rm[15:0]

else

<operand1> = Rm[31:16]

if (bit[16] == 0)

<operand2> = Rs[15:0]

else

<operand2> = Rs[31:16]

acc0[39:0] = sign_extend(<operand1> * <operand2>) + acc0[39:0]

Exceptions: none

Qualifiers Condition Code

S bit is always cleared; no condition code flags are updated

Notes: Instruction timings can be found
in ["Multiply Instruction Timings" on page 185](#).

Specifying R15 for register Rs or Rm has unpredictable results.

acc0 is defined to be 0b000 on Intel XScale processor.

The **MIAxy** instruction performs one 16-bit signed multiply and accumulates these to a single 40-bit accumulator. **x** refers to either the upper half or lower half of register Rm (multiplicand) and **y** refers to the upper or lower half of Rs (multiplier). A value of 0x1 will select bits [31:16] of the register which is specified in the mnemonic as T (for top). A value of 0x0 will select bits [15:0] of the register which is specified in the mnemonic as B (for bottom).

MIAxy does not support unsigned multiplication; all values in Rs and Rm will be interpreted as signed data values.



The instruction is only executed if the condition specified in the instruction matches the condition code status.

3.8.3.1.2 Internal Accumulator Access Format

The Intel XScale processor defines a new instruction format for accessing internal accumulators in CP0. Table 69, “Internal Accumulator Access Format” on page 173 shows that the op code falls into the coprocessor register transfer space.

The *RdHi* and *RdLo* fields allow up to 64 bits of data transfer between Intel® StrongARM® registers and an internal accumulator. The *acc* field specifies 1 of 8 internal accumulators to transfer data to/from. The Intel XScale processor implements a single 40-bit accumulator referred to as *acc0*; future implementations can specify multiple internal accumulators of varying sizes, up to 64 bits.

Access to the internal accumulator is allowed in all processor modes (user and privileged) as long as bit 0 of the Coprocessor Access Register is set. (See “Register 15: Coprocessor Access Register” on page 107 for more details).

The IXP45X/IXP46X network processors implement two instructions **MAR** and **MRA** that move two Intel® StrongARM® registers to *acc0* and move *acc0* to two Intel® StrongARM® registers, respectively.

Table 69. Internal Accumulator Access Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																					
<table><tr><td colspan="4">cond</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>L</td><td colspan="4">RdHi</td><td colspan="4">RdLo</td><td colspan="4">0</td><td>0</td><td>0</td><td>0</td><td colspan="4">0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>acc</td></tr></table>																																cond				1	1	0	0	0	1	0	L	RdHi				RdLo				0				0	0	0	0				0	0	0	0	0	acc
cond				1	1	0	0	0	1	0	L	RdHi				RdLo				0				0	0	0	0				0	0	0	0	0	acc																																
<table><tr><th>Bits</th><th>Description</th><th>Notes</th></tr><tr><td>31:28</td><td>cond - Intel® StrongARM® condition codes</td><td>-</td></tr><tr><td>20</td><td>L - move to/from internal accumulator 0= move to internal accumulator (MAR) 1= move from internal accumulator (MRA)</td><td>-</td></tr><tr><td>19:16</td><td>RdHi - specifies the high order eight (39:32) bits of the internal accumulator.</td><td>On a read of the acc, this 8-bit high order field will be sign extended. On a write to the acc, the lower 8 bits of this register will be written to acc[39:32]</td></tr><tr><td>15:12</td><td>RdLo - specifies the low order 32 bits of the internal accumulator</td><td>-</td></tr><tr><td>7:4</td><td>Should be zero</td><td></td></tr><tr><td>3</td><td>Should be zero</td><td>-</td></tr><tr><td>2:0</td><td>acc - specifies 1 of 8 internal accumulators</td><td>Intel XScale processor only implements acc0; access to any other acc is unpredictable</td></tr></table>																																Bits	Description	Notes	31:28	cond - Intel® StrongARM® condition codes	-	20	L - move to/from internal accumulator 0= move to internal accumulator (MAR) 1= move from internal accumulator (MRA)	-	19:16	RdHi - specifies the high order eight (39:32) bits of the internal accumulator.	On a read of the acc, this 8-bit high order field will be sign extended. On a write to the acc, the lower 8 bits of this register will be written to acc[39:32]	15:12	RdLo - specifies the low order 32 bits of the internal accumulator	-	7:4	Should be zero		3	Should be zero	-	2:0	acc - specifies 1 of 8 internal accumulators	Intel XScale processor only implements acc0; access to any other acc is unpredictable													
Bits	Description	Notes																																																																		
31:28	cond - Intel® StrongARM® condition codes	-																																																																		
20	L - move to/from internal accumulator 0= move to internal accumulator (MAR) 1= move from internal accumulator (MRA)	-																																																																		
19:16	RdHi - specifies the high order eight (39:32) bits of the internal accumulator.	On a read of the acc, this 8-bit high order field will be sign extended. On a write to the acc, the lower 8 bits of this register will be written to acc[39:32]																																																																		
15:12	RdLo - specifies the low order 32 bits of the internal accumulator	-																																																																		
7:4	Should be zero																																																																			
3	Should be zero	-																																																																		
2:0	acc - specifies 1 of 8 internal accumulators	Intel XScale processor only implements acc0; access to any other acc is unpredictable																																																																		

Note:

MAR has the same encoding as **MCRR** (to coprocessor 0) and **MRA** has the same encoding as **MRRC** (to coprocessor 0). These instructions move 64-bits of data to/from Intel® StrongARM® registers from/to coprocessor registers. **MCRR** and **MRRC** are defined in Intel® StrongARM®’s DSP instruction set.

Disassemblers not aware of **MAR** and **MRA** will produce the following syntax:

```
MCRR{<cond>} p0, 0x0, RdLo, RdHi, c0
```

```
MRRC{<cond>} p0, 0x0, RdLo, RdHi, c0
```


**Table 70. MAR{<cond>} acc0, RdLo, RdHi**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	0	0	0	1	0	0	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0	0	0

Operation: if ConditionPassed(<cond>) then
 acc0[39:32] = RdHi[7:0]
 acc0[31:0] = RdLo[31:0]

Exceptions: none

Qualifiers Condition Code
 No condition code flags are updated

Notes: Instruction timings can be found in
 [“Multiply Instruction Timings” on page 185](#)
 Specifying R15 as either RdHi or RdLo has unpredictable results.

The MAR instruction moves the value in register RdLo to bits[31:0] of the 40-bit accumulator (acc0) and moves bits[7:0] of the value in register RdHi into bits[39:32] of acc0.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.

Table 71. MRA{<cond>} RdLo, RdHi, acc0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	1	0	0	0	1	0	1	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0	0	0

Operation: if ConditionPassed(<cond>) then
 RdHi[31:0] = sign_extend(acc0[39:32])
 RdLo[31:0] = acc0[31:0]

Exceptions: none

Qualifiers Condition Code
 No condition code flags are updated

Notes: Instruction timings can be found in
 [“Multiply Instruction Timings” on page 185](#)
 Specifying the same register for RdHi and RdLo has unpredictable results.
 Specifying R15 as either RdHi or RdLo has unpredictable results.

The MRA instruction moves the 40-bit accumulator value (acc0) into two registers. Bits[31:0] of the value in acc0 are moved into the register RdLo. Bits[39:32] of the value in acc0 are sign extended to 32 bits and moved into the register RdHi.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.



3.8.3.2 New Page Attributes

The Intel XScale processor extends the page attributes defined by the C and B bits in the page descriptors with an additional X bit. This bit allows four more attributes to be encoded when X=1. These new encodings include allocating data for the mini-data cache and write-allocate caching. A full description of the encodings can be found in “Memory Attributes” on page 70.

The Intel XScale processor retains Intel StrongARM definitions of the C and B encoding when X = 0, which is different than the Intel StrongARM products. The memory attribute for the mini-data cache has been moved and replaced with the write-through caching attribute.

When write-allocate is enabled, a store operation that misses the data cache (cacheable data only) will generate a line fill. If disabled, a line fill only occurs when a load operation misses the data cache (cacheable data only).

Write-through caching causes all store operations to be written to memory, whether they are cacheable or not cacheable. This feature is useful for maintaining data cache coherency.

Bit 1 in the Control Register (coprocessor 15, register 1, opcode=1) is used reserved for the P bit memory attribute for memory accesses made during page table walks. The P bit is not implemented on the IXP45X/IXP46X network processors.

These attributes are programmed in the translation table descriptors, which are highlighted in:

- Table 72, “First-Level Descriptors” on page 175
- Table 73, “Second-Level Descriptors for Coarse Page Table” on page 175
- Table 74, “Second-Level Descriptors for Fine Page Table” on page 176

Two second-level descriptor formats have been defined for the IXP45X/IXP46X network processors, one is used for the coarse page table and the other is used for the fine page table.

Table 72. First-Level Descriptors

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																												0	0		
Coarse page table base address																		P	Domain				SBZ		0	1					
Section base address										SBZ			TEX		AP	P	Domain				0	C	B	1	0						
Fine page table base address																		SBZ		P	Domain				SBZ		1	1			

Table 73. Second-Level Descriptors for Coarse Page Table

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																												0	0		
Large page base address														TEX		AP3	AP2	AP1	AP0	C	B	0	1								
Small page base address																AP3	AP2	AP1	AP0	C	B	1	0								
Extended small page base address																SBZ	TEX		AP	C	B	1	1								

**Table 74. Second-Level Descriptors for Fine Page Table**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																													0	0	
Large page base address															TEX		AP3	AP2	AP1	AP0	C	B	0	1							
Small page base address											AP3	AP2	AP1	AP0	C	B	1	0													
Tiny Page Base Address											TEX				AP	C	B	1	1												

The TEX (Type Extension) field is present in several of the descriptor types. In the Intel XScale processor, only the LSB of this field is defined; this is called the X bit. The remaining bits are reserved for future use and should be programmed as zero (SBZ) on the IXP45X/IXP46X network processors.

A Small Page descriptor does not have a TEX field. For these descriptors, TEX is implicitly zero; that is, they operate as if the X bit had a '0' value.

The X bit, when set, modifies the meaning of the C and B bits. Description of page attributes and their encoding can be found in [“Memory Management Unit” on page 69](#).

3.8.3.3 Additions to CP15 Functionality

To accommodate the functionality in the Intel XScale processor, registers in CP15 and CP14 have been added or augmented. See [“Configuration” on page 96](#) for details.

At times it is necessary to be able to guarantee exactly when a CP15 update takes effect. For example, when enabling memory address translation (turning on the MMU), it is vital to know when the MMU is actually guaranteed to be in operation. To address this need, a processor-specific code sequence is defined for the Intel XScale processor. The sequence — called CPWAIT — is shown in [Example 12 on page 108](#).

Example 17. CPWAIT: Canonical Method to Wait for CP15 Update

```
;; The following macro should be used when software needs to be
;; assured that a CP15 update has taken effect.
;; It may only be used while in a privileged mode, because it
;; accesses CP15.

MACRO CPWAIT

    MRC P15, 0, R0, C2, C0, 0          ; arbitrary read of CP15
    MOV R0, R0                          ; wait for it
    SUB PC, PC, #4                      ; branch to next instruction

    ; At this point, any previous CP15 writes are
    ; guaranteed to have taken effect.

ENDM
```



When setting multiple CP15 registers, system software may opt to delay the assurance of their update. This is accomplished by executing CPWAIT only after the sequence of MCR instructions.

The CPWAIT sequence guarantees that CP15 side-effects are complete by the time the CPWAIT is complete. It is possible, however, that the CP15 side-effect will take place before CPWAIT completes or is issued. Programmers should take care that this does not affect the correctness of their code.

3.8.3.4 Event Architecture

3.8.3.4.1 Exception Summary

Table 75 shows all the exceptions that the Intel XScale processor may generate, and the attributes of each. Subsequent sections give details on each exception.

Table 75. Exception Summary

Exception Description	Exception Type ¹	Precise	Updates FAR
Reset	Reset	N	N
FIQ	FIQ	N	N
IRQ	IRQ	N	N
External Instruction	Prefetch	Y	N
Instruction MMU	Prefetch	Y	N
Instruction Cache Parity	Prefetch	Y	N
Lock Abort	Data	Y	N
MMU Data	Data	Y	Y
External Data	Data	N	N
Data Cache Parity	Data	N	N
Software Interrupt	Software Interrupt	Y	N
Undefined Instruction	Undefined Instruction	Y	N
Debug Events ²	varies	varies	N

Notes:

1. Exception types are those described in the Intel® StrongARM® manual, section 2.5.
2. Refer to “[Software Debug](#)” on page 111 for more details

3.8.3.4.2 Event Priority

The Intel XScale processor follows the exception priority specified in the ARM* *Architecture Reference Manual*. The processor has additional exceptions that might be generated while debugging. For information on these debug exceptions, see “[Software Debug](#)” on page 111.

Table 76. Event Priority

Exception	Priority
Reset	1 (Highest)
Data Abort (Precise & Imprecise)	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined Instruction, SWI	6 (Lowest)



3.8.3.4.3 Prefetch Aborts

The Intel XScale processor detects three types of prefetch aborts: Instruction MMU abort, external abort on an instruction access, and an instruction cache parity error. These aborts are described in [Table 77](#).

When a prefetch abort occurs, hardware reports the highest priority one in the extended Status field of the Fault Status Register. The value placed in R14_ABORT (the link register in abort mode) is the address of the aborted instruction + 4.

Table 77. Processors': Encoding of Fault Status for Prefetch Aborts

Priority	Sources	FS[10,3:0] [†]	Domain	FAR
Highest	Instruction MMU Exception Several exceptions can generate this encoding: - translation faults - domain faults, and - permission faults It is up to software to figure out which one occurred.	0b10000	invalid	invalid
	External Instruction Error Exception This exception occurs when the external memory system reports an error on an instruction cache fetch.	0b10110	invalid	invalid
Lowest	Instruction Cache Parity Error Exception	0b11000	invalid	invalid

[†] All other encodings not listed in the table are reserved.

3.8.3.4.4 Data Aborts

Two types of data aborts exist in the Intel XScale processor: precise and imprecise. A precise data abort is defined as one where R14_ABORT always contains the PC (+8) of the instruction that caused the exception. An imprecise abort is one where R14_ABORT contains the PC (+4) of the next instruction to execute and not the address of the instruction that caused the abort. In other words, instruction execution will have advanced beyond the instruction that caused the data abort.

On the Intel XScale processor precise data aborts are recoverable and imprecise data aborts are not recoverable.

Precise Data Aborts

- A lock abort is a precise data abort; the extended Status field of the Fault Status Register is set to 0xb10100. This abort occurs when a lock operation directed to the MMU (instruction or data) or instruction cache causes an exception, due to either a translation fault, access permission fault or external bus fault.
The Fault Address Register is undefined and R14_ABORT is the address of the aborted instruction + 8.
- A data MMU abort is precise. These are due to an alignment fault, translation fault, domain fault, permission fault or external data abort on an MMU translation. The status field is set to a predetermined Intel® StrongARM* definition which is shown in [Table 78](#).
The Fault Address Register is set to the effective data address of the instruction and R14_ABORT is the address of the aborted instruction + 8.

**Table 78. Intel XScale® Processor Encoding of Fault Status for Data Aborts**

Priorit y	Sources		FS[10,3:0] [†]	Domain	FAR
Highest	Alignment		0b000x1	invalid	valid
	External Abort on Translation	First level Second level	0b01100 0b01110	invalid valid	valid valid
	Translation	Section Page	0b00101 0b00111	invalid valid	valid valid
	Domain	Section Page	0b01001 0b01011	valid valid	valid valid
	Permission	Section Page	0b01101 0b01111	valid valid	valid valid
	Lock Abort This data abort occurs on an MMU lock operation (data or instruction TLB) or on an Instruction Cache lock operation.		0b10100	invalid	invalid
	Imprecise External Data Abort		0b10110	invalid	invalid
Lowest	Data Cache Parity Error Exception		0b11000	invalid	invalid

† All other encodings not listed in the table are reserved.

Imprecise Data Aborts

- A data cache parity error is imprecise; the extended Status field of the Fault Status Register is set to 0xb11000.
- All external data aborts except for those generated on a data MMU translation are imprecise.

The Fault Address Register for all imprecise data aborts is undefined and R14_ABORT is the address of the next instruction to execute + 4, which is the same for both Intel® StrongARM® and Thumb mode.

Although the Intel XScale processor guarantees the *Base Restored Abort Model* for precise aborts, it cannot do so in the case of imprecise aborts. A Data Abort handler may encounter an updated base register if it is invoked because of an imprecise abort.

Imprecise data aborts may create scenarios that are difficult for an abort handler to recover. Both external data aborts and data cache parity errors may result in corrupted data in the targeted registers. Because these faults are imprecise, it is possible that the corrupted data will have been used before the Data Abort fault handler is invoked. Because of this, software should treat imprecise data aborts as unrecoverable.

Note that even memory accesses marked as “stall until complete” (see [“Details on Data Cache and Write Buffer Behavior” on page 71](#)) can result in imprecise data aborts. For these types of accesses, the fault is somewhat less imprecise than the general case: it is guaranteed to be raised within three instructions of the instruction that caused it. In other words, if a “stall until complete” LD or ST instruction triggers an imprecise fault, then that fault will be seen by the program within three instructions.

With this knowledge, it is possible to write code that accesses “stall until complete” memory with impunity. Simply place several NOP instructions after such an access. If an imprecise fault occurs, it will do so during the NOPs; the data abort handler will see identical register and memory state as it would with a precise exception, and so should be able to recover. An example of this is shown in [Example 18 on page 180](#).



Example 18. Shielding Code from Potential Imprecise Aborts

```
;; Example of code that maintains architectural state through the
;; window where an imprecise fault might occur.

        LD      R0, [R1]                ; R1 points to stall-until-complete
                                           ; region of memory

        NOP

        NOP

        NOP

        ; Code beyond this point is guaranteed not to see any aborts
        ; from the LD.
```

If a system design precludes events that could cause external aborts, then such precautions are not necessary.

Multiple Data Aborts

Multiple data aborts may be detected by hardware but only the highest priority one will be reported. If the reported data abort is precise, software can correct the cause of the abort and re-execute the aborted instruction. If the lower priority abort still exists, it will be reported. Software can handle each abort separately until the instruction successfully executes.

If the reported data abort is imprecise, software needs to check the SPSR to see if the previous context was executing in abort mode. If this is the case, the link back to the current process has been lost and the data abort is unrecoverable.

3.8.3.4.5 Events from Preload Instructions

A **PLD** instruction will never cause the Data MMU to fault for any of the following reasons:

- Domain Fault
- Permission Fault
- Translation Fault

If execution of the **PLD** would cause one of the above faults, then the **PLD** causes no effect.

This feature allows software to issue **PLDs** speculatively. For example, [Example 19 on page 181](#) places a **PLD** instruction early in the loop. This **PLD** is used to fetch data for the next loop iteration. In this example, the list is terminated with a node that has a null pointer. When execution reaches the end of the list, the **PLD** on address 0x0 will not cause a fault. Rather, it will be ignored and the loop will terminate normally.



Example 19. Speculatively issuing PLD

```
;; R0 points to a node in a linked list. A node has the following layout:
;; Offset  Contents
;; -----
;;      0   data
;;      4   pointer to next node
;; This code computes the sum of all nodes in a list. The sum is placed into R9.
;;
      MOV R9, #0      ; Clear accumulator
sumList:
      LDR R1, [R0, #4] ; R1 gets pointer to next node
      LDR R3, [R0]     ; R3 gets data from current node
      PLD [R1]         ; Speculatively start load of next node
      ADD R9, R9, R3    ; Add into accumulator
      MOVS R0, R1      ; Advance to next node. At end of list?
      BNE sumList      ; If not then loop
```

3.8.3.4.6 Debug Events

Debug events are covered in “[Debug Exceptions](#)” on page 115.

3.9 Performance Considerations

This section describes relevant performance considerations that compiler writers, application programmers, and system designers need to be aware of to efficiently use the IXP45X/IXP46X network processors. Performance numbers discussed here include interrupt latency, branch prediction, and instruction latencies.

3.9.1 Interrupt Latency

Minimum Interrupt Latency is defined as the minimum number of cycles from the assertion of any interrupt signal (IRQ or FIQ) to the execution of the instruction at the vector for that interrupt. This number assumes best case conditions exist when the interrupt is asserted, e.g., the system isn’t waiting on the completion of some other operation.

A sometimes more useful number to work with is the *Maximum Interrupt Latency*. This is typically a complex calculation that depends on what else is going on in the system at the time the interrupt is asserted. Some examples that can adversely affect interrupt latency are:

- The instruction currently executing could be a 16-register LDM
- The processor could fault just when the interrupt arrives
- The processor could be waiting for data from a load, doing a page table walk, etc.
- High core-to-system (bus) clock ratios



Maximum Interrupt Latency can be reduced by:

- Ensuring that the interrupt vector and interrupt service routine are resident in the instruction cache. This can be accomplished by locking them down into the cache.
- Removing or reducing the occurrences of hardware page table walks. This also can be accomplished by locking down the application's page table entries into the TLBs, along with the page table entry for the interrupt service routine.

3.9.2 Branch Prediction

The IXP45X/IXP46X network processors implement dynamic branch prediction for the Intel® StrongARM® instructions **B** and **BL** and for the thumb instruction **B**. Any instruction that specifies the PC as the destination is predicted as not taken. For example, an **LDR** or a **MOV** that loads or moves directly to the PC will be predicted not taken and incur a branch latency penalty.

These instructions -- Intel® StrongARM® **B**, Intel® StrongARM® **BL** and thumb **B** -- enter into the branch target buffer when they are "taken" for the first time. (A "taken" branch refers to when they are evaluated to be true.) Once in the branch target buffer, the IXP45X/IXP46X network processors dynamically predict the outcome of these instructions based on previous outcomes. Table 79 shows the branch latency penalty when these instructions are correctly predicted and when they are not. A penalty of zero for correct prediction means that the IXP45X/IXP46X network processors can execute the next instruction in the program flow in the cycle following the branch.

Table 79. Branch Latency Penalty

Core Clock Cycles		Description
Intel® StrongARM®	Thumb	
+0	+ 0	Predicted Correctly. The instruction is in the branch target cache and is correctly predicted.
+4	+ 5	Mispredicted. There are three occurrences of branch misprediction, all of which incur a 4-cycle branch delay penalty. 1. The instruction is in the branch target buffer and is predicted not-taken, but is actually taken. 2. The instruction is not in the branch target buffer and is a taken branch. 3. The instruction is in the branch target buffer and is predicted taken, but is actually not-taken

3.9.3 Addressing Modes

All load and store addressing modes implemented in the IXP45X/IXP46X network processors do not add to the instruction latencies numbers.

3.9.4 Instruction Latencies

The latencies for all the instructions are shown in the following sections with respect to their functional groups: branch, data processing, multiply, status register access, load/store, semaphore, and coprocessor.

The following section explains how to read these tables.

3.9.4.1 Performance Terms

- Issue Clock (cycle 0)
The first cycle when an instruction is decoded **and** allowed to proceed to further stages in the execution pipeline (i.e., when the instruction is actually issued).



- **Cycle Distance from A to B**
The cycle distance from cycle **A** to cycle **B** is **(B-A)** -- that is, the number of cycles from the start of cycle **A** to the start of cycle **B**. Example: the cycle distance from cycle 3 to cycle 4 is one cycle.
- **Issue Latency**
The cycle distance **from** the first issue clock of the current instruction **to** the issue clock of the next instruction. The actual number of cycles can be influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.
- **Result Latency**
The cycle distance **from** the first issue clock of the current instruction **to** the issue clock of the first instruction that can use the result without incurring a resource dependency stall. The actual number of cycles can be influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.
- **Minimum Issue Latency (without Branch Misprediction)**
The minimum cycle distance **from** the issue clock of the current instruction **to** the first possible issue clock of the next instruction assuming best case conditions (i.e., that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; the current instruction does not incur resource dependency stalls during execution that cannot be detected at issue time; and if the instruction uses dynamic branch prediction, correct prediction is assumed).
- **Minimum Result Latency**
The required minimum cycle distance **from** the issue clock of the current instruction **to** the issue clock of the first instruction that can use the result without incurring a resource dependency stall assuming best case conditions (i.e., that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; and the current instruction does not incur resource dependency stalls during execution that cannot be detected at issue time).
- **Minimum Issue Latency (with Branch Misprediction)**
The minimum cycle distance **from** the issue clock of the current branching instruction **to** the first possible issue clock of the next instruction. This definition is identical to *Minimum Issue Latency* except that the branching instruction has been incorrectly predicted. It is calculated by adding *Minimum Issue Latency (without Branch Misprediction)* to the minimum branch latency penalty number from [Table 79](#), which is four cycles.
- **Minimum Resource Latency**
The minimum cycle distance from the issue clock of the current multiply instruction to the issue clock of the next multiply instruction assuming the second multiply does not incur a data dependency and is immediately available from the instruction cache or memory interface.
For the following code fragment, here is an example of computing latencies:

Example 20. Computing Latencies

UMLAL	r6, r8, r0, r1
ADD	r9, r10, r11
SUB	r2, r8, r9
MOV	r0, r1

[Table 80](#) shows how to calculate Issue Latency and Result Latency for each instruction. Looking at the issue column, the **UMLAL** instruction starts to issue on cycle 0 and the next instruction, **ADD**, issues on cycle 2, so the Issue Latency for **UMLAL** is two. From



the code fragment, there is a result dependency between the **UMLAL** instruction and the **SUB** instruction. In [Table 80](#), **UMLAL** starts to issue at cycle 0 and the **SUB** issues at cycle 5. thus the Result Latency is five.

Table 80. Latency Example

Cycle	Issue	Executing
0	umlal (1st cycle)	--
1	umlal (2nd cycle)	umlal
2	add	umlal
3	sub (stalled)	umlal & add
4	sub (stalled)	umlal
5	sub	umlal
6	mov	sub
7	--	mov

3.9.4.2 Branch Instruction Timings

Table 81. Branch Instruction Timings (Those Predicted by the BTB)

Mnemonic	Minimum Issue Latency When Correctly Predicted by the BTB	Minimum Issue Latency with Branch Misprediction
B	1	5
BL	1	5

Table 82. Branch Instruction Timings (Those not Predicted by the BTB)

Mnemonic	Minimum Issue Latency When the Branch is not Taken	Minimum Issue Latency When the Branch is Taken
BLX(1)	N/A	5
BLX(2)	1	5
BX	1	5
Data Processing Instruction with PC as the destination	Same as Table 83	4 + numbers in Table 83
LDR PC, <>	2	8
LDM with PC in register list	3 + numreg [†]	10 + max (0, numreg-3)

[†] numreg is the number of registers in the register list including the PC.

3.9.4.3 Data Processing Instruction Timings

Table 83. Data Processing Instruction Timings (Sheet 1 of 2)

Mnemonic	<shifter operand> is NOT a Shift / Rotate by Register		<shifter operand> is a Shift / Rotate by Register OR <shifter operand> is RRX	
	Minimum Issue Latency	Minimum Result Latency [†]	Minimum Issue Latency	Minimum Result Latency [†]
ADC	1	1	2	2
ADD	1	1	2	2
AND	1	1	2	2
BIC	1	1	2	2

[†] If the next instruction needs to use the result of the data processing for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

**Table 83. Data Processing Instruction Timings (Sheet 2 of 2)**

Mnemonic	<shifter operand> is NOT a Shift / Rotate by Register		<shifter operand> is a Shift / Rotate by Register OR <shifter operand> is RRX	
	Minimum Issue Latency	Minimum Result Latency [†]	Minimum Issue Latency	Minimum Result Latency [†]
CMN	1	1	2	2
CMP	1	1	2	2
EOR	1	1	2	2
MOV	1	1	2	2
MVN	1	1	2	2
ORR	1	1	2	2
RSB	1	1	2	2
RSC	1	1	2	2
SBC	1	1	2	2
SUB	1	1	2	2
TEQ	1	1	2	2
TST	1	1	2	2

† If the next instruction needs to use the result of the data processing for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.4 Multiply Instruction Timings

Table 84. Multiply Instruction Timings (Sheet 1 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency [†]	Minimum Resource Latency (Throughput)
MLA	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4
MUL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4

† If the next instruction needs to use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.



Table 84. Multiply Instruction Timings (Sheet 2 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency [†]	Minimum Resource Latency (Throughput)
SMLAL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMLALxy	N/A	N/A	2	RdLo = 2; RdHi = 3	2
SMLAWy	N/A	N/A	1	3	2
SMLAxy	N/A	N/A	1	2	1
SMULL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMULWy	N/A	N/A	1	3	2
SMULxy	N/A	N/A	1	2	1
UMLAL	Rs[31:15] = 0x00000	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
UMULL	Rs[31:15] = 0x00000	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5

[†] If the next instruction needs to use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

**Table 85. Multiply Implicit Accumulate Instruction Timings**

Mnemonic	Rs Value (Early Termination)	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MIA	Rs[31:15] = 0x0000 or Rs[31:15] = 0xFFFF	1	1	1
	Rs[31:27] = 0x0 or Rs[31:27] = 0xF	1	2	2
	all others	1	3	3
MIAxy	N/A	1	1	1
MIAPH	N/A	1	2	2

Table 86. Implicit Accumulator Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MAR	2	2	2
MRA	1	(RdLo = 2; RdHi = 3) [†]	2

[†] If the next instruction needs to use the result of the MRA for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.5 Saturated Arithmetic Instructions

Table 87. Saturated Data Processing Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
QADD	1	2
QSUB	1	2
QDADD	1	2
QDSUB	1	2

3.9.4.6 Status Register Access Instructions

Table 88. Status Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRS	1	2
MSR	2 (6 if updating mode bits)	1

3.9.4.7 Load/Store Instructions

Table 89. Load and Store Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDR	1	3 for load data; 1 for writeback of base
LDRB	1	3 for load data; 1 for writeback of base
LDRBT	1	3 for load data; 1 for writeback of base
LDRD	1 (+1 if Rd is R12)	3 for Rd; 4 for Rd+1; 1 (+1 if Rd is R12) for write-back of base
LDRH	1	3 for load data; 1 for writeback of base
LDRSB	1	3 for load data; 1 for writeback of base

**Table 89. Load and Store Instruction Timings**

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDRSH	1	3 for load data; 1 for writeback of base
LDRT	1	3 for load data; 1 for writeback of base
PLD	1	N/A
STR	1	1 for writeback of base
STRB	1	1 for writeback of base
STRBT	1	1 for writeback of base
STRD	2	2 for write-back of base
STRH	1	1 for writeback of base
STRT	1	1 for writeback of base

Table 90. Load and Store Multiple Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDM ¹	2 + numreg ²	5-18 for load data (4 + numreg for last register in list; 3 + numreg for 2nd to last register in list; 2 + numreg for all other registers in list); 2+ numreg for write-back of base
STM	2 + numreg	2 + numreg for write-back of base

Notes:

- See Table 82 on page 184 for LDM timings when R15 is in the register list.
- numreg is the number of registers in the register list.

3.9.4.8 Semaphore Instructions**Table 91. Semaphore Instruction Timings**

Mnemonic	Minimum Issue Latency	Minimum Result Latency
SWP	5	5
SWPB	5	5

3.9.4.9 Coprocessor Instructions**Table 92. CP15 Register Access Instruction Timings**

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC [†]	4	4
MCR	2	N/A

[†] MRC to R15 is unpredictable.

Table 93. CP14 Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC	8	8
MRC to R15	9	9
MCR	8	N/A
LDC	11	N/A
STC	8	N/A



3.9.4.10 Miscellaneous Instruction Timing

Table 94. Exception-Generating Instruction Timings

Mnemonic	Minimum latency to first instruction of exception handler
SWI	6
BKPT	6
UNDEFINED	6

Table 95. Count Leading Zeros Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
CLZ	1	1

3.9.4.11 Thumb Instructions

In general, the timing of Thumb instructions are the same as their equivalent Intel® StrongARM® instructions, except for the cases listed below.

- If the equivalent Intel® StrongARM® instruction maps to one in [Table 81 on page 184](#), the “Minimum Issue Latency with Branch Misprediction” goes from 5 to 6 cycles. This is due to the branch latency penalty. (See [Table 79 on page 182](#).)
- If the equivalent Intel® StrongARM® instruction maps to one in [Table 82 on page 184](#), the “Minimum Issue Latency when the Branch is Taken” increases by 1 cycle. This is due to the branch latency penalty. (See [Table 79 on page 182](#).)
- A Thumb BL instruction when H = 0 will have the same timing as an Intel® StrongARM® data processing instruction.

The mapping of Thumb instructions to Intel® StrongARM® instructions can be found in the *ARM® Architecture Reference Manual*.

3.10 Optimization Guide

3.10.1 Introduction

This document contains optimization techniques for achieving the highest performance from the IXP45X/IXP46X network processors' architecture. It is written for developers who are optimizing compilers or performance analysis tools for the devices based on these processors. It can also be used by application developers to obtain the best performance from their assembly language code. The optimizations presented in this section are based on the IXP45X/IXP46X network processors, and hence can be applied to all products that are based on it.

The IXP45X/IXP46X network processors' architecture includes a super-pipelined RISC architecture with an enhanced memory pipeline. The instruction set for the IXP45X/IXP46X network processors is based on the Intel® StrongARM® V5TE architecture; however, the IXP45X/IXP46X network processors include new instructions. Code generated for the SA110, SA1100 and SA1110 will execute on the IXP45X/IXP46X network processors, however to obtain the maximum performance of your application code, it should be optimized for the IXP45X/IXP46X network processors using the techniques presented in this document.



3.10.1.1 About This Section

This guide assumes that you are familiar with the Intel® StrongARM® instruction set and the C language. It consists of the following sections:

- “[Introduction](#)” on page 189 — Outlines the contents of this guide.
- “[Processor Pipeline](#)” on page 190 — Provides an overview of pipeline behavior for the IXP45X/IXP46X network processors.
- “[Basic Optimizations](#)” on page 195 — Outlines basic optimizations that can be applied to the IXP45X/IXP46X network processors.
- “[Cache and Prefetch Optimizations](#)” on page 201 — Contains optimizations for efficient use of caches. Also included are optimizations that take advantage of the prefetch instruction of the IXP45X/IXP46X network processors.
- “[Instruction Scheduling](#)” on page 212 — Shows how to optimally schedule code for the pipeline of the IXP45X/IXP46X network processors.
- “[Optimizing C Libraries](#)” on page 220 — Contains information relating to optimizations for C library routines.
- “[Optimizations for Size](#)” on page 220 — Contains optimizations that reduce the size of the generated code. Thumb optimizations are also included.

3.10.2 Processor Pipeline

One of the biggest differences between the IXP45X/IXP46X network processors and Intel® StrongARM® processors is the pipeline. Many of the differences are summarized in [Figure 27](#). This section provides a brief description of the structure and behavior of the IXP45X/IXP46X network processors’ pipeline.

3.10.2.1 General Pipeline Characteristics

While the pipelines for the IXP45X/IXP46X network processors are scalar and single issue, instructions may occupy all three pipelines at once. Out of order completion is possible. The following sections discuss general pipeline characteristics.

3.10.2.1.1 Number of Pipeline Stages

The IXP45X/IXP46X network processors have a longer pipeline (seven stages versus five stages) which operates at a much higher frequency than its predecessors do. This allows for greater overall performance. However, the longer pipeline has several negative consequences:

- Larger branch misprediction penalty (four cycles in the IXP45X/IXP46X network processors instead of one in Intel® StrongARM® Architecture). This is mitigated by dynamic branch prediction.
- Larger load use delay (LUD) - LUDs arise from load-use dependencies. A load-use dependency gives rise to a LUD if the result of the load instruction cannot be made available by the pipeline in due time for the subsequent instruction. An optimizing compiler should find independent instructions to fill the slot following the load.
- Certain instructions incur a few extra cycles of delay on the IXP45X/IXP46X network processors as compared to Intel® StrongARM® processors (**LDM**, **STM**).
- Decode and register file lookups are spread out over two cycles in the IXP45X/IXP46X network processors, instead of one cycle in predecessors.



3.10.2.1.2 Processor Pipeline Organization

The IXP45X/IXP46X network processors' single-issue super-pipeline consists of a main execution pipeline, MAC pipeline, and a memory access pipeline. These are shown in Figure 27, with the main execution pipeline shaded.

Figure 27. RISC Super-Pipeline

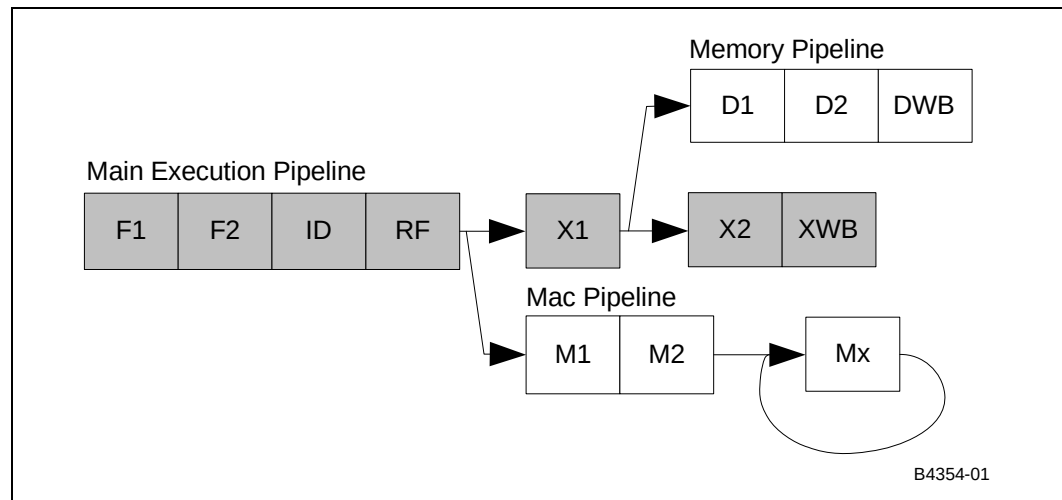


Table 96 gives a brief description of each pipe-stage.

Table 96. Pipelines and Pipe Stages

Pipe / Pipe State	Description	Covered In
Main Execution Pipeline	Handles data processing instructions	"Main Execution Pipeline" on page 193
IF1/IF2	Instruction Fetch	"
ID	Instruction Decode	"
RF	Register File / Operand Shifter	"
X1	ALU Execute	"
X2	State Execute	"
XWB	Write-back	"
Memory Pipeline	Handles load/store instructions	"Memory Pipeline" on page 194
D1/D2	Data Cache Access	"
DWB	Data cache writeback	"
MAC Pipeline	Handles all multiply instructions	"Multiply/Multiply Accumulate (MAC) Pipeline" on page 195
M1-M5	Multiplier stages	"
MWB (not shown)	MAC write-back - may occur during M2-M5	"

3.10.2.1.3 Out-of-Order Completion

Sequential consistency of instruction execution relates to two aspects: first, to the order in which the instructions are completed; and second, to the order in which memory is accessed due to load and store instructions. The IXP45X/IXP46X network processors preserve a weak processor consistency because instructions may complete out of order, provided that no data dependencies exist.



While instructions are issued in-order, the main execution pipeline, memory, and MAC pipelines are not lock-stepped, and, therefore, have different execution times. This means that instructions may finish out of program order. Short 'younger' instructions may be finished earlier than long 'older' ones. (The term 'to finish' is used here to indicate that the operation has been completed and the result has been written back to the register file.)

3.10.2.1.4 Register Scoreboarding

In certain situations, the pipeline may need to be stalled because of register dependencies between instructions. A register dependency occurs when a previous MAC or load instruction is about to modify a register value that has not been returned to the register file and the current instruction needs access to the same register. Only the destination of MAC operations and memory loads are scoreboarded. The destinations of ALU instructions are not scoreboarded.

If no register dependencies exist, the pipeline will not be stalled. For example, if a load operation has missed the data cache, subsequent instructions that do not depend on the load may complete independently.

3.10.2.1.5 Use of Bypassing

The IXP45X/IXP46X network processors' pipeline makes extensive use of bypassing to minimize data hazards. Bypassing allows results forwarding from multiple sources, eliminating the need to stall the pipeline.

3.10.2.2 Instruction Flow Through the Pipeline

The IXP45X/IXP46X network processors' pipeline issues a single instruction per clock cycle. Instruction execution begins at the F1 pipe stage and completes at the WB pipe stage.

Although a single instruction may be issued per clock cycle, all three pipelines (MAC, memory, and main execution) may be processing instructions simultaneously. If there are no data hazards, then each instruction may complete independently of the others.

Each pipe stage takes a single clock cycle or machine cycle to perform its subtask with the exception of the MAC unit.

3.10.2.2.1 Intel® StrongARM® V5TE Instruction Execution

Figure 27 on page 191 uses arrows to show the possible flow of instructions in the pipeline. Instruction execution flows from the F1 pipe stage to the RF pipe stage. The RF pipe stage may issue a single instruction to either the X1 pipe stage or the MAC unit (multiply instructions go to the MAC, while all others continue to X1). This means that M1 or X1 will be idle.

All load/store instructions are routed to the memory pipeline after the effective addresses have been calculated in X1.

The Intel StrongARM V5TE bx (branch and exchange) instruction, which is used to branch between Intel StrongARM and thumb code, causes the entire pipeline to be flushed (The bx instruction is not dynamically predicted by the BTB). If the processor is in Thumb mode, then the ID pipe stage dynamically expands each Thumb instruction into a normal Intel StrongARM V5TE RISC instruction and execution resumes as usual.



3.10.2.2.2 Pipeline Stalls

The progress of an instruction can stall anywhere in the pipeline. Several pipe stages may stall for various reasons. It is important to understand when and how hazards occur in the IXP45X/IXP46X network processors' pipeline. Performance degradation can be significant if care is not taken to minimize pipeline stalls.

3.10.2.3 Main Execution Pipeline

3.10.2.3.1 F1 / F2 (Instruction Fetch) Pipe Stages

The job of the instruction fetch stages F1 and F2 is to present the next instruction to be executed to the ID stage. Several important functional units reside within the F1 and F2 stages, including:

- Branch Target Buffer (BTB)
- Instruction Fetch Unit (IFU)

An understanding of the BTB (See [“Branch Target Buffer” on page 83](#)) and IFU are important for performance considerations. A summary of operation is provided here so that the reader may understand its role in the F1 pipe stage.

- Branch Target Buffer (BTB)
The BTB predicts the outcome of branch type instructions. Once a branch type instruction reaches the X1 pipe stage, its target address is known. If this address is different from the address that the BTB predicted, the pipeline is flushed, execution starts at the new target address, and the branch's history is updated in the BTB.
- Instruction Fetch Unit (IFU)
The IFU is responsible for delivering instructions to the *instruction decode* (ID) pipe stage. One instruction word is delivered each cycle (if possible) to the ID. The instruction could come from one of two sources: instruction cache or fetch buffers.

3.10.2.3.2 ID (Instruction Decode) Pipe Stage

The ID pipe stage accepts an instruction word from the IFU and sends register decode information to the RF pipe stage. The ID is able to accept a new instruction word from the IFU on every clock cycle in which there is no stall. The ID pipe stage is responsible for:

- General instruction decoding (extracting the op code, operand addresses, destination addresses and the offset).
- Detecting undefined instructions and generating an exception.
- Dynamic expansion of complex instructions into sequence of simple instructions. Complex instructions are defined as ones that take more than one clock cycle to issue, such as **LDM**, **STM**, and **SWP**.

3.10.2.3.3 RF (Register File / Shifter) Pipe Stage

The main function of the RF pipe stage is to read and write to the *register file unit*, or *RFU*. It provides source data to:

- EX for ALU operations
- MAC for multiply operations
- Data Cache for memory writes
- Coprocessor interface



The ID unit decodes the instruction and specifies which registers are accessed in the RFU. Based upon this information, the RFU determines if it needs to stall the pipeline due to a register dependency. A register dependency occurs when a previous instruction is about to modify a register value that has not been returned to the RFU and the current instruction needs to access that same register. If no dependencies exist, the RFU will select the appropriate data from the register file and pass it to the next pipe stage. When a register dependency does exist, the RFU will keep track of which register is unavailable and when the result is returned, the RFU will stop stalling the pipe.

The Intel StrongARM architecture specifies that one of the operands for data processing instructions as the shifter operand, where a 32-bit shift can be performed before it is used as an input to the ALU. This shifter is located in the second half of the RF pipe stage.

3.10.2.3.4 X1 (Execute) Pipe Stages

The X1 pipe stage performs the following functions:

- ALU calculation - the ALU performs arithmetic and logic operations, as required for data processing instructions and load/store index calculations.
- Determine conditional instruction execution - The instruction's condition is compared to the CPSR prior to execution of each instruction. Any instruction with a false condition is cancelled, and will not cause any architectural state changes, including modifications of registers, memory, and PSR.
- Branch target determination - If a branch was incorrectly predicted by the BTB, the X1 pipe stage flushes all of the instructions in the previous pipe stages and sends the branch target address to the BTB, which will restart the pipeline

3.10.2.3.5 X2 (Execute 2) Pipe Stages

The X2 pipe stage contains the *program status registers* (PSRs). This pipe stage selects what is going to be written to the RFU in the WB cycle: PSRs (MRS instruction), ALU output, or other items.

3.10.2.3.6 WB (Write-Back)

When an instruction has reached the write-back stage, it is considered complete. Changes are written to the RFU.

3.10.2.4 Memory Pipeline

The memory pipeline consists of two stages, D1 and D2. The *data cache unit*, or DCU, consists of the data-cache array, mini-data cache, fill buffers, and write buffers. The memory pipeline handles load / store instructions.

3.10.2.4.1 D1 and D2 Pipe Stage

Operation begins in D1 after the X1 pipe stage has calculated the effective address for load/stores. The data cache and mini-data cache returns the destination data in the D2 pipe stage. Before data is returned in the D2 pipe stage, sign extension and byte alignment occurs for byte and half-word loads.



3.10.2.5 Multiply / Multiply Accumulate (MAC) Pipeline

The Multiply-Accumulate (MAC) unit executes the multiply and multiply-accumulate instructions supported by the IXP45X/IXP46X network processors. The MAC implements the 40-bit accumulator register (acc0) on the IXP45X/IXP46X network processors and handles the instructions, which transfer its value to and from general-purpose Intel StrongARM registers.

The following are important characteristics about the MAC:

- The MAC is not truly pipelined, as the processing of a single instruction may require use of the same data path resources for several cycles before a new instruction can be accepted. The type of instruction and source arguments determines the number of cycles required.
- No more than two instructions can occupy the MAC pipeline concurrently.
- When the MAC is processing an instruction, another instruction may not enter M1 unless the original instruction completes in the next cycle.
- The MAC unit can operate on 16-bit packed signed data. This reduces register pressure and memory traffic size. Two 16-bit data items can be loaded into a register with one LDR.
- The MAC can achieve throughput of one multiply per cycle when performing a 16-by-32-bit multiply.

3.10.2.5.1 Behavioral Description

The execution of the MAC unit starts at the beginning of the M1 pipe stage, where it receives two 32-bit source operands. Results are completed N cycles later (where N is dependent on the operand size) and returned to the register file. For more information on MAC instruction latencies, refer to [“Instruction Latencies” on page 182](#).

An instruction that occupies the M1 or M2 pipe stages will also occupy the X1 and X2 pipe stage, respectively. Each cycle, a MAC operation progresses for M1 to M5. A MAC operation may complete anywhere from M2-M5. If a MAC operation enters M3-M5, it is considered committed because it will modify architectural state regardless of subsequent events.

3.10.3 Basic Optimizations

This section outlines optimizations specific to Intel StrongARM architecture. These optimizations have been modified to suit the IXP45X/IXP46X network processors where needed.

3.10.3.1 Conditional Instructions

The IXP45X/IXP46X network processors' architecture provides the ability to execute instructions conditionally. This feature combined with the ability of the IXP45X/IXP46X network processors instructions to modify the condition codes makes possible a wide array of optimizations.

3.10.3.1.1 Optimizing Condition Checks

The IXP45X/IXP46X network processors' instructions can selectively modify the state of the condition codes. When generating code for if-else and loop conditions it is often beneficial to make use of this feature to set condition codes, thereby eliminating the need for a subsequent compare instruction.

Consider the C code segment:



```
if (a + b)
```

Code generated for the if condition without using an add instruction to set condition codes is:

```
;Assume r0 contains the value a, and r1 contains the value b
    add    r0,r0,r1
    cmp    r0, #0
```

However, code can be optimized as follows making use of add instruction to set condition codes:

```
;Assume r0 contains the value a, and r1 contains the value b
    adds   r0,r0,r1
```

The instructions that increment or decrement the loop counter can also be used to modify the condition codes. This eliminates the need for a subsequent compare instruction. A conditional branch instruction can then be used to exit or continue with the next loop iteration.

Consider the following C code segment:

```
for (i = 10; i != 0; i--)
{
    do something;
}
```

The optimized code generated for the above code segment would look like:

```
L6:
.
.
    subs r3, r3, #1
    bne  .L6
```

It is also beneficial to rewrite loops whenever possible so as to make the loop exit conditions check against the value 0. For example, the code generated for the code segment below will need a compare instruction to check for the loop exit condition.

```
for (i = 0; i < 10; i++)
{
    do something;
}
```

If the loop were rewritten as follows, the code generated avoids using the compare instruction to check for the loop exit condition.

```
for (i = 9; i >= 0; i--)
{
    do something;
}
```

3.10.3.1.2 Optimizing Branches

Branches decrease application performance by indirectly causing pipeline stalls. Branch prediction improves the performance by lessening the delay inherent in fetching a new instruction stream. The number of branches that can accurately be predicted is limited by the size of the branch target buffer. Since the total number of branches executed in



a program is relatively large compared to the size of the branch target buffer; it is often beneficial to minimize the number of branches in a program. Consider the following C code segment.

```
int foo(int a)
{
    if (a > 10)
        return 0;
    else
        return 1;
}
```

The code generated for the if-else portion of this code segment using branches is:

```
    cmp    r0, #10
    ble    L1
    mov    r0, #0
    b      L2
L1:
    mov    r0, #1
L2:
```

The code generated above takes three cycles to execute the else part and four cycles for the if-part assuming best case conditions and no branch misprediction penalties. In the case of the IXP45X/IXP46X network processors, a branch misprediction incurs a penalty of four cycles. If the branch is incorrectly predicted 50 percent of the time, and if we assume that both the if-part and the else-part are equally likely to be taken, on an average the code above takes 5.5 cycles to execute.

$$\left(\frac{50}{100} \times 4 + \frac{3+4}{2} \right) = 5.5 \quad \text{cycles}$$

If we were to use the IXP45X/IXP46X network processors to execute instructions conditionally, the code generated for the above if-else statement is:

```
    cmp    r0, #10
    movgt  r0, #0
    movle  r0, #1
```

The above code segment would not incur any branch misprediction penalties and would take three cycles to execute assuming best case conditions. As can be seen, using conditional instructions speeds up execution significantly. However, the use of conditional instructions should be carefully considered to ensure that it does improve performance. To decide when to use conditional instructions over branches consider the following hypothetical code segment:



```

if (cond)
    if_stmt
else
    else_stmt

```

Assume that we have the following data:

- $N1_B$ Number of cycles to execute the if_stmt assuming the use of branch instructions
- $N2_B$ Number of cycles to execute the else_stmt assuming the use of branch instructions
- $P1$ Percentage of times the if_stmt is likely to be executed
- $P2$ Percentage of times we are likely to incur a branch misprediction penalty
- $N1_C$... Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be true
- $N2_C$... Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be false

Once we have the above data, use conditional instructions when:

$$\left(N1_C \times \frac{P1}{100}\right) + \left(N2_C \times \frac{100 - P1}{100}\right) \leq \left(N1_B \times \frac{P1}{100}\right) + \left(N2_B \times \frac{100 - P1}{100}\right) + \left(\frac{P2}{100} \times 4\right)$$

The following example illustrates a situation in which we are better off using branches over conditional instructions. Consider the code sample shown below:

```

cmp    r0, #0
bne    L1
add    r0, r0, #1
add    r1, r1, #1
add    r2, r2, #1
add    r3, r3, #1
add    r4, r4, #1
b      L2
L1:
sub    r0, r0, #1
sub    r1, r1, #1
sub    r2, r2, #1
sub    r3, r3, #1
sub    r4, r4, #1
L2:

```

In the above code sample, the cmp instruction takes 1 cycle to execute, the if-part takes 7 cycles to execute and the else-part takes 6 cycles to execute. If we were to change the code above so as to eliminate the branch instructions by making use of conditional instructions, the if-else part would always take 10 cycles to complete.

If we make the assumptions that both paths are equally likely to be taken and that branches are mis-predicted 50% of the time, the costs of using conditional execution Vs using branches can be computed as follows:

Cost of using conditional instructions:



$$1 + \left(\frac{50}{100} \times 10\right) + \left(\frac{50}{100} \times 10\right) = 11 \quad \text{cycles}$$

Cost of using branches:

$$1 + \left(\frac{50}{100} \times 7\right) + \left(\frac{50}{100} \times 6\right) + \left(\frac{50}{100} \times 4\right) = 9.5 \quad \text{cycles}$$

As can be seen, we get better performance by using branch instructions in the above scenario.

3.10.3.1.3 Optimizing Complex Expressions

Conditional instructions should also be used to improve the code generated for complex expressions such as the C shortcut evaluation feature. Consider the following C code segment:

```
int foo(int a, int b)
{
    if (a != 0 && b != 0)
        return 0;
    else
        return 1;
}
```

The optimized code for the if condition is:

```
cmp    r0, #0
cmpne  r1, #0
```

Similarly, the code generated for the following C segment

```
int foo(int a, int b)
{
    if (a != 0 || b != 0)
        return 0;
    else
        return 1;
}
```

is:

```
cmp    r0, #0
cmpeq  r1, #0
```

The use of conditional instructions in the above fashion improves performance by minimizing the number of branches, thereby minimizing the penalties caused by branch incorrect predictions. This approach also reduces the utilization of branch prediction resources.

3.10.3.2 Bit Field Manipulation

The shift and logical operations of the IXP45X/IXP46X network processors provide a useful way of manipulating bit fields. Bit field operations can be optimized as follows:



```
;Set the bit number specified by r1 in register r0
mov   r2, #1
orr   r0, r0, r2, asl r1
;Clear the bit number specified by r1 in register r0
mov   r2, #1
bic   r0, r0, r2, asl r1
;Extract the bit-value of the bit number specified by r1 of the
;value in r0 storing the value in r0
mov   r1, r0, asr r1
and   r0, r1, #1
;Extract the higher order 8 bits of the value in r0 storing
;the result in r1
mov   r1, r0, lsr #24
```

3.10.3.3 Optimizing the Use of Immediate Values

The **MOV** or **MVN** instruction of the IXP45X/IXP46X network processors should be used when loading an immediate (constant) value into a register. Please refer to the *ARM* Architecture Reference Manual* for the set of immediate values that can be used in a **MOV** or **MVN** instruction. It is also possible to generate a whole set of constant values using a combination of **MOV**, **MVN**, **ORR**, **BIC**, and **ADD** instructions. The **LDR** instruction has the potential of incurring a cache miss in addition to polluting the data and instruction caches. The code samples below illustrate cases when a combination of the above instructions can be used to set a register to a constant value:

```
;Set the value of r0 to 127
mov   r0, #127
;Set the value of r0 to 0xfffffebf.
mvn   r0, #260
;Set the value of r0 to 257
mov   r0, #1
orr   r0, r0, #256
;Set the value of r0 to 0x51f
mov   r0, #0x1f
orr   r0, r0, #0x500
;Set the value of r0 to 0xf100ffff
mvn   r0, #0xff, 16
bic   r0, r0, #0xe, 8
; Set the value of r0 to 0x12341234
mov   r0, #0x8d, 30
orr   r0, r0, #0x1, 20
add   r0, r0, r0, LSL #16 ; shifter delay of 1 cycle
```

Note that it is possible to load any 32-bit value into a register using a sequence of four instructions.

3.10.3.4 Optimizing Integer Multiply and Divide

Multiplication by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Multiplication of R0 by 2n
mov   r0, r0, LSL #n
;Multiplication of R0 by 2n+1
add   r0, r0, r0, LSL #n
```

Multiplication by an integer constant that can be expressed as $(2^n + 1) \cdot (2^m)$ can similarly be optimized as:



```
;Multiplication of r0 by an integer constant that can be
;expressed as (2n+1)*(2m)
    add    r0, r0, r0, LSL #n
    mov    r0, r0, LSL #m
```

Please note that the above optimization should only be used in cases where the multiply operation cannot be advanced far enough to prevent pipeline stalls.

Dividing an unsigned integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing an unsigned value by an integer constant
;that can be represented as 2n
    mov    r0, r0, LSR #n
```

Dividing a signed integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing a signed value by an integer constant
;that can be represented as 2n
    mov    r1, r0, ASR #31
    add    r0, r0, r1, LSR #(32 - n)
    mov    r0, r0, ASR #n
```

The add instruction would stall for 1 cycle. The stall can be prevented by filling in another instruction before add.

3.10.3.5 Effective Use of Addressing Modes

The IXP45X/IXP46X network processors provide a variety of addressing modes that make indexing an array of objects highly efficient. For a detailed description of these addressing modes please refer to the *ARM* Architecture Reference Manual*. The following code samples illustrate how various kinds of array operations can be optimized to make use of these addressing modes:

```
;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the next word
    str    r1, [r0], #4
;Increment the contents of r0 to make it point to the next word
;and set the contents of the word pointed to the value contained
;in r1
    str    r1, [r0, #4]!
;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the previous word
    str    r1, [r0], #-4
;Decrement the contents of r0 to make it point to the previous
;word and set the contents of the word pointed to the value
;contained in r1
    str    r1, [r0, #-4]!
```

3.10.4 Cache and Prefetch Optimizations

This section considers how to use the various cache memories in all their modes and then examines when and how to use prefetch to improve execution efficiencies.

3.10.4.1 Instruction Cache

The IXP45X/IXP46X network processors have separate instruction and data caches. Only fetched instructions are held in the instruction cache even though both data and instructions may reside within the same memory space with each other. Functionally,



the instruction cache is either enabled or disabled. There is no performance benefit in not using the instruction cache. The exception is that code, which locks code into the instruction cache, must itself execute from non-cached memory.

3.10.4.1.1 Cache Miss Cost

The performance of the IXP45X/IXP46X network processors is highly dependent on reducing the cache miss rate.

Note that this cycle penalty becomes significant when the core is running much faster than external memory. Executing non-cached instructions severely curtails the processor's performance in this case and it is very important to do everything possible to minimize cache misses.

For the IXP45X/IXP46X network processors, care must be taken to optimize code to have a maximum cache hit when accesses have been requested to the Expansion Bus Interface or the PCI Bus Controller. These design recommendations are due to the latency that may be associated with accessing the PCI Bus Controller and Expansion Bus Controller. Retries will be issued to the Intel XScale processor until the requested transaction is completed.

3.10.4.1.2 Round Robin Replacement Cache Policy

Both the data and the instruction caches use a round robin replacement policy to evict a cache line. The simple consequence of this is that at sometime every line will be evicted, assuming a non-trivial program. The less obvious consequence is that predicting when and over which cache lines evictions take place is very difficult to predict. This information must be gained by experimentation using performance profiling.

3.10.4.1.3 Code Placement to Reduce Cache Misses

Code placement can greatly affect cache misses. One way to view the cache is to think of it as 32 sets of 32 bytes, which span an address range of 1,024 bytes. When running, the code maps into 32 blocks modular 1,024 of cache space. Any sets, which are overused, will thrash the cache. The ideal situation is for the software tools to distribute the code on a temporal evenness over this space.

This is very difficult if not impossible for a compiler to do. Most of the input needed to best estimate how to distribute the code will come from profiling followed by compiler based two pass optimizations.

3.10.4.1.4 Locking Code into the Instruction Cache

One very important instruction cache feature is the ability to lock code into the instruction cache. Once locked into the instruction cache, the code is always available for fast execution. Another reason for locking critical code into cache is that with the round robin replacement policy, eventually the code will be evicted, even if it is a very frequently executed function. Key code components to consider for locking are:

- Interrupt handlers
- Real time clock handlers
- OS critical code
- Time critical application code

The disadvantage to locking code into the cache is that it reduces the cache size for the rest of the program. How much code to lock is very application dependent and requires experimentation to optimize.



Code placed into the instruction cache should be aligned on a 1,024-byte boundary and placed sequentially together as tightly as possible so as not to waste precious memory space. Making the code sequential also insures even distribution across all cache ways. Though it is possible to choose randomly located functions for cache locking, this approach runs the risk of landing multiple cache ways in one set and few or none in another set. This distribution unevenness can lead to excessive thrashing of the Data and Mini Caches.

3.10.4.2 Data and Mini Cache

The IXP45X/IXP46X network processors allow the user to define memory regions whose cache policies can be set by the user (see ["Cacheability" on page 88](#)). Supported policies and configurations are:

- Non Cacheable with no coalescing of memory writes.
- Non Cacheable with coalescing of memory writes.
- Mini-Data cache with write coalescing, read allocate, and write-back caching.
- Mini-Data cache with write coalescing, read allocate, and write-through caching.
- Mini-Data cache with write coalescing, read-write allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-through caching.
- Data cache with write coalescing, read-write allocate, and write-back caching.

To support allocating variables to these various memory regions, the tool chain (compiler, assembler, linker and debugger), must implement named sections.

The performance of your application code depends on what cache policy you are using for data objects. A description of when to use a particular policy is described below.

The IXP45X/IXP46X network processors allow dynamic modification of the cache policies at run time, however, the operation is requires considerable processing time and therefore should not be used by applications.

If the application is running under an OS, then the OS may restrict you from using certain cache policies.

3.10.4.2.1 Non-Cacheable Regions

It is recommended that non-cache memory (X=0, C=0, and B=0) be used only if necessary as is often necessary for I/O devices. Accessing non-cacheable memory is likely to cause the processor to stall frequently due to the long latency of memory reads.

3.10.4.2.2 Write-Through and Write-Back Cached Memory Regions

Write through memory regions generate more data traffic on the bus. Therefore is not recommended that the write-through policy be used. The write back policy must be used whenever possible.

However, in a multiprocessor environment it will be necessary to use a write through policy if data is shared across multiple processors. In such a situation all shared memory regions should use write through policy. Memory regions that are private to a particular processor should use the write back policy.



3.10.4.2.3 Read Allocate and Read-Write Allocate Memory Regions

Most of the regular data and the stack for your application should be allocated to a read-write allocate region. It is expected that you will be writing and reading from them often.

Data that is write only (or data that is written to and subsequently not used for a long time) should be placed in a read allocate region. Under the read-allocate policy if a cache write miss occurs a new cache line will not be allocated, and hence will not evict critical data from the Data cache.

3.10.4.2.4 Creating On-Chip RAM

Part of the Data cache can be converted into fast on chip RAM. Access to objects in the on-chip RAM will not incur cache miss penalties, thereby reducing the number of processor stalls. Application performance can be improved by converting a part of the cache into on chip RAM and allocating frequently allocated variables to it. Due to the round-robin replacement policy of the IXP45X/IXP46X network processors, all data will eventually be evicted. Therefore to prevent critical or frequently used data from being evicted it should be allocated to on-chip RAM.

The following variables are good candidates for allocating to the on-chip RAM:

- Frequently used global data used for storing context for context switching.
- Global variables that are accessed in time critical functions such as interrupt service routines.

The on-chip RAM is created by locking a memory region into the Data cache (see [“Reconfiguring the Data Cache as Data RAM” on page 92](#) for more details).

When creating the on-chip RAM, care must be taken to ensure that all sets in the on-chip RAM area of the Data cache have approximately the same number of ways locked, otherwise some sets will have more ways locked than the others. This uneven allocation will increase the level of thrashing in some sets and leave other sets under utilized.

For example, consider three arrays arr1, arr2 and arr3 of size 64 bytes each that are being allocated to the on-chip RAM and assume that the address of arr1 is 0, address of arr2 is 1024, and the address of arr3 is 2048. All three arrays will be within the same sets, i.e. set0 and set1, as a result three ways in both sets set0 and set1, will be locked, leaving 29 ways for use by other variables.

This can be overcome by allocating on-chip RAM data in sequential order. In the above example allocating arr2 to address 64 and arr3 to address 128, allows the three arrays to use only 1 way in sets 0 through 5.

3.10.4.2.5 Mini-Data Cache

The mini-data cache is best used for data structures, which have short temporal lives, and/or cover vast amounts of data space. Addressing these types of data spaces from the Data cache would corrupt much if not all of the Data cache by evicting valuable data. Eviction of valuable data will reduce performance. Placing this data instead in Mini-data cache memory region would prevent Data cache corruption while providing the benefits of cached accesses.

A prime example of using the mini-data cache would be for caching the procedure call stack. The stack can be allocated to the mini-data cache so that its use does not trash the main dcache. This would keep local variables from global data.

Following are examples of data that could be assigned to mini-dcache:



- The stack space of a frequently occurring interrupt, the stack is used only during the duration of the interrupt, which is usually very small.
- Video buffers, these are usual large and can occupy the whole cache.

Over use of the Mini-Data cache will thrash the cache. This is easy to do because the Mini-Data cache only has two ways per set. For example, a loop which uses a simple statement such as:

```
for (i=0; i< IMAX; i++)
{
    A[i] = B[i] + C[i];
}
```

Where A, B, and C reside in a mini-data cache memory region and each is array is aligned on a 1-K boundary will quickly thrash the cache.

3.10.4.2.6 Data Alignment

Cache lines begin on 32-byte address boundaries. To maximize cache line use and minimize cache pollution, data structures should be aligned on 32-byte boundaries and sized to multiple cache line sizes. Aligning data structures on cache address boundaries simplifies later addition of prefetch instructions to optimize performance.

Not aligning data on cache lines has the disadvantage of moving the prefetch address correspondingly to the misalignment. Consider the following example:

```
struct {
    long ia;
    long ib;
    long ic;
    long id;
} tdata[IMAX];

for (i=0, i<IMAX; i++)
{
    PREFETCH(tdata[i+1].ia);
    tdata[i].ia = tdata[i].ib + tdata[i].ic + tdata[i].id;
    ....
    tdata[i].id = 0;
}
```

In this case if tdata[] is not aligned to a cache line, then the prefetch using the address of tdata[i+1].ia may not include element id. If the array was aligned on a cache line + 12 bytes, then the prefetch would have to be placed on &tdata[i+1].id.

If the structure is not sized to a multiple of the cache line size, then the prefetch address must be advanced appropriately and will require extra prefetch instructions. Consider the following example:



```
struct {
    long ia;
    long ib;
    long ic;
    long id;
    long ie;
} tdata[IMAX];

ADDRESS preadd = tdata

for (i=0, i<IMAX; i++)
{
    PREFETCH(predadd+=16);
    tdata[I].ia = tdata[I].ib + tdata[I].ic + tdata[I].id +
    tdata[I].ie;
    ....
    tdata[I].ie = 0;
}
```

In this case, the prefetch address was advanced by size of half a cache line and every other prefetch instruction is ignored. Further, an additional register is required to track the next prefetch address.

Generally, not aligning and sizing data will add extra computational overhead.

Additional prefetch considerations are discussed in greater detail in following sections.

3.10.4.2.7 Literal Pools

The IXP45X/IXP46X network processors do not have a single instruction that can move all literals (a constant or address) to a register. One technique to load registers with literals in the IXP45X/IXP46X network processors is by loading the literal from a memory location that has been initialized with the constant or address. These blocks of constants are referred to as literal pools. See [“Basic Optimizations” on page 195](#) for more information on how to do this. It is advantageous to place all the literals together in a pool of memory known as a literal pool. These data blocks are located in the text or code address space so that they can be loaded using PC relative addressing. However, references to the literal pool area load the data into the data cache instead of the instruction cache. Therefore it is possible that the literal may be present in both the data and instruction caches, resulting in waste of space.

For maximum efficiency, the compiler should align all literal pools on cache boundaries and size each pool to a multiple of 32 bytes (the size of a cache line). One additional optimization would be group highly used literal pool references into the same cache line. The advantage is that once one of the literals has been loaded, the other seven will be available immediately from the data cache.

3.10.4.3 Cache Considerations

3.10.4.3.1 Cache Conflicts, Pollution, and Pressure

Cache pollution occurs when unused data is loaded in the cache and cache pressure occurs when data that is not temporal to the current process is loaded into the cache.

3.10.4.3.2 Memory Page Thrashing

Memory page thrashing occurs because of the nature of DDRI SDRAM. DDRI SDRAMs are typically divided into four banks. Each bank can have one selected page where a page address size for current memory components is often defined as 4 k. Memory lookup time or latency time for a selected page address is currently two to three bus clocks. Thrashing occurs when subsequent memory accesses within the same memory



bank access different pages. The memory page change adds three to four bus clock cycles to memory latency. This added delay extends the prefetch distance correspondingly making it more difficult to hide memory access latencies. This type of thrashing can be resolved by placing the conflicting data structures into different memory banks or by paralleling the data structures such that the data resides within the same memory page. It is also extremely important to insure that instruction and data sections are in different memory banks, or they will continually trash the memory page selection.

3.10.4.4 Prefetch Considerations

The IXP45X/IXP46X network processors have a true prefetch load instruction (PLD). The purpose of this instruction is to preload data into the data and mini-data caches. Data prefetching allows hiding of memory transfer latency while the processor continues to execute instructions. The prefetch is important to compiler and assembly code because judicious use of the prefetch instruction can enormously improve throughput performance of the IXP45X/IXP46X network processors. Data prefetch can be applied not only to loops but also to any data references within a block of code. Prefetch also applies to data writing when the memory type is enabled as write-allocate.

The prefetch load instruction of the IXP45X/IXP46X network processors is a true prefetch instruction because the load destination is the data or mini-data cache and not a register. Compilers for processors which have data caches, but do not support prefetch, sometimes use a load instruction to preload the data cache. This technique has the disadvantages of using a register to load data and requiring additional registers for subsequent preloads and thus increasing register pressure. By contrast, the prefetch can be used to reduce register pressure instead of increasing it.

The prefetch load is a hint instruction and does not guarantee that the data will be loaded. Whenever the load would cause a fault or a table walk, then the processor will ignore the prefetch instruction, the fault or table walk, and continue processing the next instruction. This is particularly advantageous in the case where a linked list or recursive data structure is terminated by a NULL pointer. Prefetching the NULL pointer will not fault program flow.

3.10.4.4.1 Prefetch Loop Limitations

It is not always advantages to add prefetch to a loop. Loop characteristics that limit the use value of prefetch are discussed below.

3.10.4.4.2 Compute versus Data Bus Bound

At the extreme, a loop, which is data bus bound, will not benefit from prefetch because all the system resources to transfer data are quickly allocated and there are no instructions that can profitably be executed. On the other end of the scale, compute bound loops allow complete hiding of all data transfer latencies.

3.10.4.4.3 Low Number of Iterations

Loops with very low iteration counts may have the advantages of prefetch completely mitigated. A loop with a small fixed number of iterations may be faster if the loop is completely unrolled rather than trying to schedule prefetch instructions.

3.10.4.4.4 Bandwidth Limitations

Overuse of prefetches can usurp resources and degrade performance. This happens because once the bus traffic requests exceed the system resource capacity, the processor stalls. The data transfer resources for the IXP45X/IXP46X network processors are:



- Four fill buffers
- Four pending buffers
- Eight half-cache line write buffer

DDR1 SDRAM resources are typically:

- Four memory banks
- One page buffer per bank referencing a 4K address range
- Four transfer request buffers

Consider how these resources work together. A fill buffer is allocated for each cache read miss. A fill buffer is also allocated each cache write miss if the memory space is write allocate along with a pending buffer. A subsequent read to the same cache line does not require a new fill buffer, but does require a pending buffer and a subsequent write will also require a new pending buffer. A fill buffer is also allocated for each read to a non-cached memory and a write buffer is needed for each memory write to non-cached memory that is non-coalescing. Consequently, a **STM** instruction listing eight registers and referencing non-cached memory will use eight write buffers assuming they don't coalesce and two write buffers if they do coalesce. A cache eviction requires a write buffer for each dirty bit set in the cache line. The prefetch instruction requires a fill buffer for each cache line and 0, 1, or 2 write buffers for an eviction.

When adding prefetch instructions, caution must be asserted to insure that the combination of prefetch and instruction bus requests do not exceed the system resource capacity described above or performance will be degraded instead of improved. The important points are to spread prefetch operations over calculations so as to allow bus traffic to free flow and to minimize the number of necessary prefetches.

3.10.4.4.5 Cache Memory Considerations

Stride, the way data structures are walked through, can affect the temporal quality of the data and reduce or increase cache conflicts. The data cache and mini-data caches for the IXP45X/IXP46X network processors each have 32 sets of 32 bytes. This means that each cache line in a set is on a modular 1-K-address boundary. The caution is to choose data structure sizes and stride requirements that do not overwhelm a given set causing conflicts and increased register pressure. Register pressure can be increased because additional registers are required to track prefetch addresses. The effects can be affected by rearranging data structure components to use more parallel access to search and compare elements. Similarly rearranging sections of data structures so that sections often written fit in the same half cache line, 16 bytes for the IXP45X/IXP46X network processors, can reduce cache eviction write-backs. On a global scale, techniques such as array merging can enhance the spatial locality of the data.

As an example of array merging, consider the following code:

```
int a_array[NMAX];
int b_array[NMAX];
int ix;

for (i=0; i<NMAX; i++)
{
    ix = b[i];
    if (a[i] != 0)
        ix = a[i];
    do_other_calculations;
}
```

In the above code, data is read from both arrays a and b, but a and b are not spatially close. Array merging can place a and b specially close.



```

struct {
    int a;
    int b;
} c_arrays;

int ix;

for (i=0; i<NMAX; i++)
{
    ix = c[i].b;
    if (c[i].a != 0)
        ix = c[i].a;
    do_other_calculations;
}

```

As an example of rearranging often written to sections in a structure, consider the code sample:

```

struct employee {
    struct employee *prev;
    struct employee *next;
    float Year2DatePay;
    float Year2DateTax;
    int ssno;
    int empid;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};

```

In the data structure shown above, the fields Year2DatePay, Year2DateTax, Year2Date401KDed, and Year2DateOtherDed are likely to change with each pay check. The remaining fields however change very rarely. If the fields are laid out as shown above, assuming that the structure is aligned on a 32-byte boundary, modifications to the Year2Date fields is likely to use two write buffers when the data is written out to memory. However, we can restrict the number of write buffers that are commonly used to 1 by rearranging the fields in the above data structure as shown below:

```

struct employee {
    struct employee *prev;
    struct employee *next;
    int ssno;
    int empid;
    float Year2DatePay;
    float Year2DateTax;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};

```

3.10.4.4.6 Cache Blocking

Cache blocking techniques, such as strip-mining, are used to improve temporal locality of the data. Given a large data set that can be reused across multiple passes of a loop, data blocking divides the data into smaller chunks which can be loaded into the cache during the first loop and then be available for processing on subsequent loops thus minimizing cache misses and reducing bus traffic.

As an example of cache blocking consider the following code:

```

for(i=0; i<10000; i++)
    for(j=0; j<10000; j++)
        for(k=0; k<10000; k++)
            C[j][k] += A[i][k] * B[j][i];

```



The variable $A[i][k]$ is completely reused. However, accessing $C[j][k]$ in the j and k loops can displace $A[i][j]$ from the cache. Using blocking the code becomes:

```
for(i=0; i<10000; i++)
    for(j1=0; j<100; j++)
        for(k1=0; k<100; k++)
            for(j2=0; j<100; j++)
                for(k2=0; k<100; k++)
                {
                    j = j1 * 100 + j2;
                    k = k1 * 100 + k2;
                    C[j][k] += A[i][k] * B[j][i];
                }
```

3.10.4.4.7 Prefetch Unrolling

When iterating through a loop, data transfer latency can be hidden by prefetching ahead one or more iterations. The solution incurs an unwanted side effect that the final interactions of a loop loads useless data into the cache, polluting the cache, increasing bus traffic and possibly evicting valuable temporal data. This problem can be resolved by prefetch unrolling. For example consider:

```
for(i=0; i<NMAX; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
```

Interactions $i-1$ and i , will prefetch superfluous data. The problem can be avoid by unrolling the end of the loop.

```
for(i=0; i<NMAX-2; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
sum += data[NMAX-2];
sum += data[NMAX-1];
```

Unfortunately, prefetch loop unrolling does not work on loops with indeterminate iterations.

3.10.4.4.8 Pointer Prefetch

Not all looping constructs contain induction variables. However, prefetching techniques can still be applied. Consider the following linked list traversal example:

```
while(p) {
    do_something(p->data);
    p = p->next;
}
```

The pointer variable p becomes a pseudo induction variable and the data pointed to by $p->next$ can be pre-fetched to reduce data transfer latency for the next iteration of the loop. Linked lists should be converted to arrays as much as possible.

```
while(p) {
    prefetch(p->next);
    do_something(p->data);
    p = p->next;
}
```



Recursive data structure traversal is another construct where prefetching can be applied. This is similar to linked list traversal. Consider the following pre-order traversal of a binary tree:

```
preorder(treeNode *t) {
    if(t) {
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

The pointer variable *t* becomes the pseudo induction variable in a recursive loop. The data structures pointed to by the values *t->left* and *t->right* can be pre-fetched for the next iteration of the loop.

```
preorder(treeNode *t) {
    if(t) {
        prefetch(t->right);
        prefetch(t->left);
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

Note the order reversal of the prefetches in relationship to the usage. If there is a cache conflict and data is evicted from the cache then only the data from the first prefetch is lost.

3.10.4.4.9 Loop Interchange

As mentioned earlier, the sequence in which data is accessed affects cache thrashing. Usually, it is best to access data in a contiguous spatially address range. However, arrays of data may have been laid out such that indexed elements are not physically next to each other. Consider the following C code which places array elements in row major order.

```
for(j=0; j<NMAX; j++)
    for(i=0; i<NMAX; i++)
    {
        prefetch(A[i+1][j]);
        sum += A[i][j];
    }
```

In the above example, *A[i][j]* and *A[i+1][j]* are not sequentially next to each other. This situation causes an increase in bus traffic when prefetching loop data. In some cases where the loop mathematics are unaffected, the problem can be resolved by induction variable interchange. The above examples becomes:

```
for(i=0; i<NMAX; i++)
    for(j=0; j<NMAX; j++)
    {
        prefetch(A[i][j+1]);
        sum += A[i][j];
    }
```

3.10.4.4.10 Loop Fusion

Loop fusion is a process of combining multiple loops, which reuse the same data, in to one loop. The advantage of this is that the reused data is immediately accessible from the data cache. Consider the following example:



```
for(i=0; i<NMAX; i++)
{
    prefetch(A[i+1], c[i+1], c[i+1]);
    A[i] = b[i] + c[i];
}
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], c[i+1], A[i+1]);
    D[i] = A[i] + c[i];
}
```

The second loop reuses the data elements A[i] and c[i]. Fusing the loops together produces:

```
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], A[i+1], c[i+1], b[i+1]);
    ai = b[i] + c[i];
    A[i] = ai;
    D[i] = ai + c[i];
}
```

3.10.4.4.11 Prefetch to Reduce Register Pressure

Pre-fetch can be used to reduce register pressure. When data is needed for an operation, then the load is scheduled far enough in advance to hide the load latency. However, the load ties up the receiving register until the data can be used. For example:

```
ldr    r2, [r0]
; Process code { not yet cached latency > 60 core clocks }
add    r1, r1, r2
```

In the above case, r2 is unavailable for processing until the add statement. Prefetching the data load frees the register for use. The example code becomes:

```
pld    [r0] ;prefetch the data keeping r2 available for use
; Process code
ldr    r2, [r0]
; Process code { ldr result latency is 3 core clocks }
add    r1, r1, r2
```

With the added prefetch, register r2 can be used for other operations until almost just before it is needed.

3.10.5 Instruction Scheduling

This section discusses instruction scheduling optimizations. Instruction scheduling refers to the rearrangement of a sequence of instructions for the purpose of minimizing pipeline stalls. Reducing the number of pipeline stalls improves application performance. While making this rearrangement, care should be taken to ensure that the rearranged sequence of instructions has the same effect as the original sequence of instructions.

3.10.5.1 Scheduling Loads

On the IXP45X/IXP46X network processors, an **LDR** instruction has a result latency of three cycles assuming the data being loaded is in the data cache. If the instruction after the **LDR** needs to use the result of the load, then it would stall for 2 cycles. If possible, the instructions surrounding the **LDR** instruction should be rearranged.



to avoid this stall. Consider the following example:

```
add    r1, r2, r3
ldr     r0, [r5]
add     r6, r0, r1
sub     r8, r2, r3
mul     r9, r2, r3
```

In the code shown above, the **ADD** instruction following the **LDR** would stall for two cycles because it uses the result of the load. The code can be rearranged as follows to prevent the stalls:

```
ldr     r0, [r5]
add     r1, r2, r3
sub     r8, r2, r3
add     r6, r0, r1
mul     r9, r2, r3
```

Note that this rearrangement may not be always possible. Consider the following example:

```
cmp     r1, #0
addne   r4, r5, #4
subeq   r4, r5, #4
ldr     r0, [r4]
cmp     r0, #10
```

In the example above, the **LDR** instruction cannot be moved before the **ADDNE** or the **SUBEQ** instructions because the **LDR** instruction depends on the result of these instructions. Rewrite the above code to make it run faster at the expense of increasing code size:

```
cmp     r1, #0
ldrne   r0, [r5, #4]
ldreq   r0, [r5, #-4]
addne   r4, r5, #4
subeq   r4, r5, #4
cmp     r0, #10
```

The optimized code takes six cycles to execute compared to the seven cycles taken by the unoptimized version.

The result latency for an **LDR** instruction is significantly higher if the data being loaded is not in the data cache. To minimize the number of pipeline stalls in such a situation the **LDR** instruction should be moved as far away as possible from the instruction that uses result of the load. Note that this may at times cause certain register values to be



spilled to memory due to the increase in register pressure. In such cases, use a preload instruction or a preload hint to ensure that the data access in the **LDR** instruction hits the cache when it executes. A preload hint should be used in cases where we cannot be sure whether the load instruction would be executed. A preload instruction should be used in cases where we can be sure that the load instruction would be executed. Consider the following code sample:

```
; all other registers are in use
sub    r1, r6, r7
mul    r3, r6, r2
mov    r2, r2, LSL #2
orr    r9, r9, #0xf
add    r0, r4, r5
ldr    r6, [r0]
add    r8, r6, r8
add    r8, r8, #4
orr    r8, r8, #0xf
; The value in register r6 is not used after this
```

In the code sample above, the **ADD** and the **LDR** instruction can be moved before the **MOV** instruction. Note that this would prevent pipeline stalls if the load hits the data cache. However, if the load is likely to miss the data cache, move the **LDR** instruction so that it executes as early as possible - before the **SUB** instruction. However, moving the **LDR** instruction before the **SUB** instruction would change the program semantics. It is possible to move the **ADD** and the **LDR** instructions before the **SUB** instruction if we allow the contents of the register r6 to be spilled and restored from the stack as shown below:

```
; all other registers are in use
str    r6, [sp, #-4]!
add    r0, r4, r5
ldr    r6, [r0]
mov    r2, r2, LSL #2
orr    r9, r9, #0xf
add    r8, r6, r8
ldr    r6, [sp], #4
add    r8, r8, #4
orr    r8, r8, #0xf
sub    r1, r6, r7
mul    r3, r6, r2
; The value in register r6 is not used after this
```

As can be seen above, the contents of the register r6 have been spilled to the stack and subsequently loaded back to the register r6 to retain the program semantics. Another way to optimize the code above is with the use of the preload instruction as shown below:

```
; all other registers are in use
add    r0, r4, r5
pld    [r0]
sub    r1, r6, r7
mul    r3, r6, r2
mov    r2, r2, LSL #2
orr    r9, r9, #0xf
ldr    r6, [r0]
add    r8, r6, r8
add    r8, r8, #4
orr    r8, r8, #0xf
; The value in register r6 is not used after this
```

The IXP45X/IXP46X network processors have four fill-buffers that are used to fetch data from external memory when a data-cache miss occurs. The IXP45X/IXP46X network processors stall when all fill buffers are in use. This happens when more than 4



loads are outstanding and are being fetched from memory. As a result, the code written should ensure that no more than four loads are outstanding at the same time. For example, the number of loads issued sequentially should not exceed four. Also note that a preload instruction may cause a fill buffer to be used. As a result, the number of preload instructions outstanding should also be considered to arrive at the number of loads that are outstanding.

Similarly, the number of write buffers also limits the number of successive writes that can be issued before the processor stalls. No more than eight stores can be issued. Also note that if the data caches are using the write-allocate with write-back policy, then a load operation may cause stores to the external memory if the read operation evicts a cache line that is dirty (modified). The number of sequential stores may be limited by this fact.

3.10.5.1.1 Scheduling Load and Store Double (LDRD/STRD)

The IXP45X/IXP46X network processors introduce two new double word instructions: **LDRD** and **STRD**. **LDRD** loads 64 bits of data from an effective address into two consecutive registers, conversely, **STRD** stores 64 bits from two consecutive registers to an effective address. There are two important restrictions on how these instructions may be used:

- The effective address must be aligned on an 8-byte boundary
- The specified register must be even (r0, r2, etc.).

If this situation occurs, using **LDRD/STRD** instead of **LDM/STM** to do the same thing is more efficient because **LDRD/STRD** issues in only one/two clock cycle(s), as opposed to **LDM/STM** which issues in four clock cycles. Avoid **LDRD**s targeting R12; this incurs an extra cycle of issue latency.

The **LDRD** instruction has a result latency of 3 or 4 cycles depending on the destination register being accessed (assuming the data being loaded is in the data cache).

```
add    r6, r7, r8
sub    r5, r6, r9
; The following ldrd instruction would load values
; into registers r0 and r1
ldrd   r0, [r3]
orr    r8, r1, #0xf
mul    r7, r0, r7
```

In the code example above, the **ORR** instruction would stall for three cycles because of the four cycle result latency for the second destination register of an **LDRD** instruction. The code shown above can be rearranged to remove the pipeline stalls:

```
; The following ldrd instruction would load values
; into registers r0 and r1
ldrd   r0, [r3]
add    r6, r7, r8
sub    r5, r6, r9
mul    r7, r0, r7
orr    r8, r1, #0xf
```

Any memory operation following a **LDRD** instruction (**LDR**, **LDRD**, **STR** and so on) would stall for 1 cycle.

```
; The str instruction below would stall for 1 cycle
ldrd   r0, [r3]
str    r4, [r5]
```



3.10.5.1.2 Scheduling Load and Store Multiple (LDM/STM)

LDM and **STM** instructions have an issue latency of 2-20 cycles depending on the number of registers being loaded or stored. The issue latency is typically two cycles plus an additional cycle for each of the registers being loaded or stored assuming a data cache hit. The instruction following an LDM would stall whether or not this instruction depends on the results of the load. A **LDRD** or **STRD** instruction does not suffer from this drawback (except when followed by a memory operation) and should be used where possible. Consider the task of adding two 64-bit integer values. Assume that the addresses of these values are aligned on an 8-byte boundary. This can be achieved using the **LDM** instructions as shown below:

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
ldm r0, {r2, r3}
ldm r1, {r4, r5}
adds r0, r2, r4
adc r1, r3, r5
```

If the code were written as shown above, assuming all the accesses hit the cache, the code would take 11 cycles to complete. Rewriting the code as shown below using **LDRD** instruction would take only seven cycles to complete. The performance would increase further if we can fill in other instructions after **LDRD** to reduce the stalls due to the result latencies of the **LDRD** instructions.

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
ldrdr2, [r0]
ldrdr4, [r1]
adds r0, r2, r4
adc r1, r3, r5
```

Similarly, the code sequence shown below takes five cycles to complete.

```
stm r0, {r2, r3}
add r1, r1, #1
```

The alternative version which is shown below would only take three cycles to complete.

```
strdr2, [r0]
add r1, r1, #1
```

3.10.5.2 Scheduling Data Processing Instructions

Most data processing instructions for the IXP45X/IXP46X network processors have a result latency of one cycle. This means that the current instruction is able to use the result from the previous data processing instruction. However, the result latency is two cycles if the current instruction needs to use the result of the previous data processing instruction for a shift by immediate. As a result, the following code segment would incur a one-cycle stall for the MOV instruction:

```
sub r6, r7, r8
add r1, r2, r3
mov r4, r1, LSL #2
```

The code above can be rearranged as follows to remove the one-cycle stall:

```
add r1, r2, r3
sub r6, r7, r8
mov r4, r1, LSL #2
```



All data processing instructions incur a two cycle issue penalty and a two-cycle result penalty when the shifter operand is a shift/rotate by a register or shifter operand is RRX. Since the next instruction would always incur a 2 cycle issue penalty, there is no way to avoid such a stall except by re-writing the assembler instruction. Consider the following segment of code:

```
mov    r3, #10
mul    r4, r2, r3
add    r5, r6, r2, LSL r3
sub    r7, r8, r2
```

The subtract instruction would incur a one-cycle stall due to the issue latency of the add instruction as the shifter operand is shift by a register. The issue latency can be avoided by changing the code as follows:

```
mov    r3, #10
mul    r4, r2, r3
add    r5, r6, r2, LSL #10
sub    r7, r8, r2
```

3.10.5.3 Scheduling Multiply Instructions

Multiply instructions can cause pipeline stalls due to either resource conflicts or result latencies. The following code segment would incur a stall of zero to three cycles depending on the values in registers r1, r2, r4 and r5 due to resource conflicts.

```
mul    r0, r1, r2
mul    r3, r4, r5
```

The following code segment would incur a stall of one to three cycles, depending on the values in registers r1 and r2 due to result latency.

```
mul    r0, r1, r2
mov    r4, r0
```

Note that a multiply instruction that sets the condition codes blocks the whole pipeline. A four-cycle multiply operation that sets the condition codes behaves the same as a 4 cycle issue operation. Consider the following code segment:

```
muls   r0, r1, r2
add    r3, r3, #1
sub    r4, r4, #1
sub    r5, r5, #1
```

The add operation above would stall for three cycles if the multiply takes four cycles to complete. It is better to replace the code segment above with the following sequence:

```
mul    r0, r1, r2
add    r3, r3, #1
sub    r4, r4, #1
sub    r5, r5, #1
cmp    r0, #0
```

Please refer to [“Instruction Latencies” on page 182](#) to get the instruction latencies for various multiply instructions. The multiply instructions should be scheduled taking into consideration these instruction latencies.



3.10.5.4 Scheduling SWP and SWPB Instructions

The **SWP** and **SWPB** instructions have a five-cycle issue latency. As a result of this latency, the instruction following the **SWP/SWPB** instruction would stall for 4 cycles. **SWP** and **SWPB** instructions should, therefore, be used only where absolutely needed.

For example, the following code may be used to swap the contents of two memory locations:

```
; Swap the contents of memory locations pointed to by r0 and r1
ldr    r2, [r0]
swp    r2, [r1]
str    r2, [r1]
```

The code above takes nine cycles to complete. The rewritten code below, takes six cycles to execute:

```
; Swap the contents of memory locations pointed to by r0 and r1
ldr    r2, [r0]
ldr    r3, [r1]
str    r2, [r1]
str    r3, [r0]
```

3.10.5.5 Scheduling the MRA and MAR Instructions (MRRC/MCRR)

The **MRA** (**MRRC**) instruction has an issue latency of one cycle, a result latency of two or three cycles depending on the destination register value being accessed and a resource latency of two cycles.

Consider the code sample:

```
mra    r6, r7, acc0
mra    r8, r9, acc0
add    r1, r1, #1
```

The code shown above would incur a one-cycle stall due to the two-cycle resource latency of an **MRA** instruction. The code can be rearranged as shown below to prevent this stall.

```
mra    r6, r7, acc0
add    r1, r1, #1
mra    r8, r9, acc0
```

Similarly, the code shown below would incur a two-cycle penalty due to the three-cycle result latency for the second destination register.

```
mra    r6, r7, acc0
mov    r1, r7
mov    r0, r6
add    r2, r2, #1
```

The stalls incurred by the code shown above can be prevented by rearranging the code:

```
mra    r6, r7, acc0
add    r2, r2, #1
mov    r0, r6
mov    r1, r7
```



The **MAR (MCRR)** instruction has an issue latency, a result latency, and a resource latency of two cycles. Due to the two-cycle issue latency, the pipeline would always stall for one cycle following a **MAR** instruction. The use of the **MAR** instruction should, therefore, be used only where absolutely necessary.

3.10.5.6 Scheduling the MIA and MIAPH Instructions

The **MIA** instruction has an issue latency of one cycle. The result and resource latency can vary from one to three cycles depending on the values in the source register.

Consider the following code sample:

```
mia  acc0, r2, r3
mia  acc0, r4, r5
```

The second **MIA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle resource latency.

Similarly, consider the following code sample:

```
mia  acc0, r2, r3
mra  r4, r5, acc0
```

The **MRA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle result latency.

The **MIAPH** instruction has an issue latency of one cycle, result latency of two cycles and a resource latency of two cycles.

Consider the code sample shown below:

```
add  r1, r2, r3
miaph acc0, r3, r4
miaph acc0, r5, r6
mra  r6, r7, acc0
sub  r8, r3, r4
```

The second **MIAPH** instruction would stall for one-cycle due to a two-cycle resource latency. The **MRA** instruction would stall for one-cycle due to a two-cycle result latency. These stalls can be avoided by rearranging the code as follows:

```
miaph acc0, r3, r4
add  r1, r2, r3
miaph acc0, r5, r6
sub  r8, r3, r4
mra  r6, r7, acc0
```

3.10.5.7 Scheduling MRS and MSR Instructions

The **MRS** instruction has an issue latency of one cycle and a result latency of two cycles. The **MSR** instruction has an issue latency of 2 cycles (6 if updating the mode bits) and a result latency of one cycle.

Consider the code sample:

```
mrs  r0, cpsr
orr  r0, r0, #1
add  r1, r2, r3
```



The **ORR** instruction above would incur a one cycle stall due to the two-cycle result latency of the **MRS** instruction. In the code example above, the **ADD** instruction can be moved before the **ORR** instruction to prevent this stall.

3.10.5.8 Scheduling CP15 Coprocessor Instructions

The **MRC** instruction has an issue latency of one cycle and a result latency of three cycles. The **MCR** instruction has an issue latency of one cycle.

Consider the code sample:

```
add    r1, r2, r3
mrc    p15, 0, r7, C1, C0, 0
mov    r0, r7
add    r1, r1, #1
```

The **MOV** instruction above would incur a two-cycle latency due to the three-cycle result latency of the **mrc** instruction. The code shown above can be rearranged as follows to avoid these stalls:

```
mrc    p15, 0, r7, C1, C0, 0
add    r1, r2, r3
add    r1, r1, #1
mov    r0, r7
```

3.10.6 Optimizing C Libraries

Many of the standard C library routines can benefit greatly by being optimized for the IXP45X/IXP46X network processors' architecture. The following string and memory manipulation routines should be tuned to obtain the best performance from the architecture (instruction selection, cache usage and data prefetch):

strcat, strchr, strcmp, strcoll, strcpy, strcspn, strlen, strncat, strncmp, strpbrk, strrchr, strspn, strstr, strtok, strxfrm, memchr, memcpy, memcpy, memmove, memset

3.10.7 Optimizations for Size

For applications such as cell phone software it is necessary to optimize the code for improved performance while minimizing code size. Optimizing for smaller code size will, in general, lower the performance of your application. This section contains techniques for optimizing for code size using the instruction set for the IXP45X/IXP46X network processors.

3.10.7.1 Space / Performance Trade Off

Many optimizations mentioned in the previous sections improve the performance of Intel® StrongARM® code. However, using these instructions will result in increased code size. Use the following optimizations to reduce the space requirements of the application code.

3.10.7.1.1 Multiple Word Load and Store

The **LDM/STM** instructions are one word long and let you load or store multiple registers at once. Use the **LDM/STM** instructions instead of a sequence of loads/stores to consecutive addresses in memory whenever possible.



3.10.7.1.2 Use of Conditional Instructions

Using conditional instructions to expand if-then-else statements as described in [“Conditional Instructions” on page 195](#) will result in increasing the size of the generated code. Therefore, do not use conditional instructions if application code space requirements are an issue.

3.10.7.1.3 Use of PLD Instructions

The preload instruction **PLD** is only a hint, it does not change the architectural state of the processor. Using or not using them will not change the behavior of your code, therefore, you should avoid using these instructions when optimizing for space.

§ §



4.0 Network Processor Engines (NPE)

The Network Processor Engines (NPE) are dedicated function processors containing hardware coprocessors that are integrated into the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The NPEs are used to offload processing functions required by the Intel XScale® Processor.

The NPEs are high-performance, hardware multi-threaded processors with additional local hardware assist functionality used to off load processor intensive functions such as MII (MAC), CRC checking/generation, AAL2, DES, AES, SHA, MD5, etc.

Note: Certain NPEs are not available, depending on which variant of the IXP45X/IXP46X network processors is used. [Table 97](#) shows which network-processor models have which NPEs enabled.

Table 97. Processors: Network Processor Functions

Device	NPEA				NPEB				NPEC	
	UTOPIA	HSS	HDLC	ETHA	ETHB1	ETHB2	ETHB3	ETHB4	ETHC	AES/DES/SHA/MD5
Intel® IXP465	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Intel® IXP460	NO	NO	NO	NO	YES	NO	NO	NO	YES	NO
Intel® IXP455	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES

All instruction code for the NPEs is stored locally through a dedicated instruction memory bus and dedicated data memory bus. The NPE's support processing of dedicated peripherals interfaces on the IXP45X/IXP46X network processors. The peripherals supported by the use of the NPEs are up to three SMII interfaces, up to three MII interfaces, UTOPIA Level 2 interface, and two high-speed serial interfaces.

The NPE core is a hardware multi-threaded processor engine that is used to accelerate functions that are difficult to achieve high performance in a standard RISC processor. Each NPE core is a 133-MHz processor core that has self-contained instruction memory and self-contained data memory that operate in parallel.

In addition to having separate instruction/data memory, the NPE core supports hardware multi-threading with support for multiple contexts. The support of hardware multi-threading allows an efficient processor engine with minimal processor stalls due to the ability of the processor core to switch context to a new context in a single clock cycle based upon a prioritized/preemptive basis.

The prioritized/preemptive nature of the context switching allows time critical applications to be implemented in a low-latency fashion, which is required when processing multi-media applications. The NPE core also connects several hardware-based coprocessors. The coprocessors are used to implement several functions that are difficult for a processor to implement. The type of functions implemented by the coprocessors are serialization/de-serialization, CRC checking/generation, DES/3DES, AES, SHA, MD-5, and HDLC bit-stuffing/de-stuffing. These coprocessors are implemented in hardware, therefore allowing the coprocessors and the Network Processor Engine core to operate in parallel.



The combined forces of the hardware multi-threading, independent instruction memory, independent data memory, and parallel processing allows the Intel XScale processor to be utilized for application purposes. The multi-processing capability of the peripheral interface functions allows unparalleled performance to be achieved by the application running on the Intel XScale processor.

Note: All the described NPE functions require Intel-supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.



5.0 Internal Buses

The internal bus architecture of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors is designed to allow parallel processing to occur and isolate bus utilization based upon particular traffic patterns. The bus is segmented into four major buses, MPI, the North AHB, the South AHB, and the APB.

The Memory Port Interface provides a dedicated interface between the Intel XScale® Processor and the DDR1 SDRAM. This interface is a 133-MHz, 64-bit bus that is mastered by the Intel XScale processor only. The only target of this interface is the DDR1 SDRAM controller.

By providing a dedicated path which supports multiple outstanding queues, split, and posted transactions, this interface can achieve better memory performance over previous generation products. Isolation of the interface also allows more DDR1 SDRAM performance to be allocated to the Intel XScale processor.

The North AHB is a 133-MHz, 32-bit bus that can be mastered by the NPE A Network Processor Engine (NPE), NPE B, or NPE C. The targets of the North AHB can be the DDR1 SDRAM controller or the AHB/AHB Bridge. The AHB/AHB Bridge will allow access by the NPEs to the peripherals and internal targets on the South AHB.

Data transfers by the NPEs from the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB Bridge may be “posted” when writing or “split” when reading — allowing control of the North AHB to be given to another master on the North AHB and allowing the bus to achieve maximum efficiency.

Transfers to the AHB/AHB Bridge are considered to be small and infrequent, relative to the traffic passed between the NPEs on the North AHB and the DDR1 SDRAM.

The South AHB is a 133-MHz, 32-bit bus that can be mastered by the Intel XScale processor, PCI Controller, Expansion Bus Controller, USB Host 2.0, and the AHB/AHB Bridge. The targets of the South AHB can be the DDR1 SDRAM, PCI Controller, Queue Manager, Expansion Bus Controller, USB Host 2.0, Cryptography Unit, or the AHB/APB Bridge. Accessing across the AHB/APB Bridge allows interfacing to peripherals attached to the APB Bus.

The APB is a 66.66 MHz, 32-bit bus that can be mastered by the AHB/APB Bridge only. The targets of the APB can be the High-Speed UART Interface, Console UART Interface, USB 1.1 interface, all NPEs, the Internal Bus Performance Monitoring Unit (PMU), Interrupt Controller, GPIO, IEEE 1588 hardware assist unit, SSP, I²C, and Timers. The APB interface to the NPEs is used for NPE code download, part configuration, and status collection.

The maximum length that any AHB master can hold the AHB is for eight 32-bit words. This feature allows for fairness among all masters on the AHBs.



5.1 Internal Bus Arbiters

The IXP45X/IXP46X network processors contain two internal bus arbiters, one arbiter for North AHB transactions and one arbiter for South AHB transactions. The arbiters are used to ensure that at any particular time only one AHB master has access to a given AHB. The arbiters perform this function by observing all of the AHB master requests to the given AHB segment and deciding which AHB master will be the next owner of the AHB.

The arbiters have a standard interface to all bus masters and split-capable slaves in the system. Any AHB master can request an AHB at any cycle. The arbiters sample the AHB requests. If the particular AHB master is requesting the AHB and is next in the round-robin list, the arbiter will grant the master the AHB.

The arbiters also have the capability to handle split transfers. A split transfer is when:

- An AHB master request a read from a split capable AHB target
- The split-capable AHB target issues a split transfer indication to the arbiter
- The arbiter allows other transactions to take place on the AHB while the AHB master that issued the request that resulted in the split transfer waits on the read data to be returned from the split capable AHB target
- The split capable AHB target completes the read transaction and notifies the arbiter
- The arbiter will grant the AHB master that requested the split transfer the bus in the normal round-robin progression
- The read data will be transferred from the split capable AHB target to the AHB master that issued the request that resulted in the split transfer

All split capable AHB targets split a single AHB master read request at any given instance. If the split capable AHB target receives another read request while servicing a split transaction, the split capable AHB target will issue a retry. The only split capable targets on the South AHB is the Expansion Bus. The only split capable target on the North AHB is the AHB/AHB Bridge.

The arbiters send event information to the Performance Monitoring Unit (PMU) so that the North AHB and South AHB performance can be observed. The events that may be monitored are provided in the chapter “Performance Monitoring Unit,” in the second volume of this two-volume component specification.

The North AHB Arbiter is identical to the South AHB Arbiter in all respects except for the bus masters and targets in which they are connected.

5.1.1 Priority Mechanism

The arbiters allow the bus initiators access to the AHBs using a round-robin scheme. [Table 98](#) illustrates a generic arbitration example for three AHB masters requesting the AHB. The functionality of the independent arbiters is identical.

Each of the bus initiators (X, Y, and Z) is constantly requesting the bus. The bottom row of [Table 98](#) lists the current bus initiator/winner of the initiators. For example, when all three masters are requesting access, X will be the winner, and then Y and Z will be requesting. Next, Y wins the AHB and X returns with a new request. So ZX are still valid with Z being the oldest. Next, Z wins the bus, etc.

**Table 98. Bus Arbitration Example: Three Requesting Masters**

	Initial	+1	+2	+3	+4	+5	+6	+7	+8	+9
Requesting Masters	XYZ	YZ	ZX	XY	YZ	ZX	XY	YZ	ZX	XY
Winning Bus Initiator	-	X	Y	Z	X	Y	Z	X	Y	Z

5.2 Memory Map

Table 99 shows the memory map of peripherals connected to the AHB.

Table 99. Memory Map (Sheet 1 of 2)

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 MB	Expansion Bus Data (Mirrored) ¹
0000_0000	3FFF_FFFF	1024 MB	DDR1-266 SDRAM Data ¹
4000_0000	47FF_FFFF	128 MB	(Reserved)
4800_0000	4FFF_FFFF	128 MB	PCI Data
5000_0000	5FFF_FFFF	256 MB	Expansion Bus Data
6000_0000	63FF_FFFF	64 MB	Queue manager
6400_0000	BFFF_FFFF		(Reserved)
C000_0000	C3FF_FFFF	64 MB	PCI Controller Configuration and Status Registers
C400_0000	C7FF_FFFF	64 MB	Expansion Bus Configuration Registers ²
C800_0000	C800_0FFF	1 KB	UART0
C800_1000	C800_1FFF	1 KB	UART1
C800_2000	C800_2FFF	1 KB	Internal Bus Performance Monitoring Unit
C800_3000	C800_3FFF	1 KB	Interrupt Controller
C800_4000	C800_4FFF	1 KB	GPIO Controller
C800_5000	C800_5FFF	1 KB	Timers
C800_6000	C800_6FFF	1 KB	NPE-A (Intel® IXP400 Software Definition)— Not User Programmable
C800_7000	C800_7FFF	1 KB	NPE-B (IXP400 software Definition) – Not User Programmable

Notes:

- The lowest 256 Mbyte of address space is configurable based on the value of a configuration register located in the Expansion Bus Controller.
- When bit 31 (MEM_MAP) of configuration register #0 (EXP_CFG0) is set to logic 1, the Expansion Bus occupies the lowest 256 Mbyte of address space.
 - When bit 31 (MEM_MAP) of configuration register #0 (EXP_CFG0) is set to logic 0 the Expansion Bus occupies 256MB of address space starting at 5000_0000 while the DDR1 SDRAM occupies the lower 256 Mbyte of address.
 - In both cases, regardless of the value of MEM_MAP, the DDR1 SDRAM occupies the 768 MB (from 1000_0000 to 3FFF_FFFF) immediately following the lowest 256 Mbyte and the Expansion Bus can be accessed starting at address 5000_0000.
 - On reset, the configuration register in the Expansion Bus will be set to logic 1. This setting is required because the dedicated boot memory is flash memory located on the Expansion Bus.

**Table 99. Memory Map (Sheet 2 of 2)**

Start Address	End Address	Size	Use
C800_8000	C800_8FFF	1 KB	NPE-C (IXP400 software Definition) – Not User Programmable
C800_9000	C800_9FFF	1 KB	Ethernet MAC 0 on NPE B
C800_A000	C800_AFFF	1 KB	Ethernet MAC on NPE C
C800_B000	C800_BFFF	1 KB	USB Device Controller
C800_C000	C800_CFFF	1KB	Ethernet MAC on NPE A
C800_D000	C800_DFFF	1KB	Ethernet MAC 1 on NPE B
C800_E000	C800_EFFF	1KB	Ethernet MAC 2 on NPE B
C800_F000	C800_FFFF	1KB	Ethernet MAC 3 on NPE B
C801_0000	C801_0FFF	1KB	IEEE1588 Hardware Assist
C801_1000	C801_1FFF	1KB	I ² C Unit
C801_2000	C801_2FFF	1KB	SSP Unit
C801_3000	CC00_E4FF		(Reserved)
CC00_E500	CC00_F5FF	1KB	DDR1-266 SDRAM Configuration Registers
CD00_0000	CDFF_FFFF	256MB	USB Host Controller
CE00_0000	FFFF_FFFF		(Reserved)

Notes:

- The lowest 256 Mbyte of address space is configurable based on the value of a configuration register located in the Expansion Bus Controller.
- When bit 31 (MEM_MAP) of configuration register #0 (EXP_CFG0) is set to logic 1, the Expansion Bus occupies the lowest 256 Mbyte of address space.
 - When bit 31 (MEM_MAP) of configuration register #0 (EXP_CFG0) is set to logic 0 the Expansion Bus occupies 256MB of address space starting at 5000_0000 while the DDR1 SDRAM occupies the lower 256 Mbyte of address.
 - In both cases, regardless of the value of MEM_MAP, the DDR1 SDRAM occupies the 768 MB (from 1000_0000 to 3FFF_FFFF) immediately following the lowest 256 Mbyte and the Expansion Bus can be accessed starting at address 5000_0000.
 - On reset, the configuration register in the Expansion Bus will be set to logic 1. This setting is required because the dedicated boot memory is flash memory located on the Expansion Bus.





6.0 Ethernet MACs

The functionality supported by the MII Interfaces is tightly coupled with the code written on the Network Processor Engine (NPE) core. This chapter details the full hardware capabilities of the MII Interface contained within the Ethernet coprocessor of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described below.

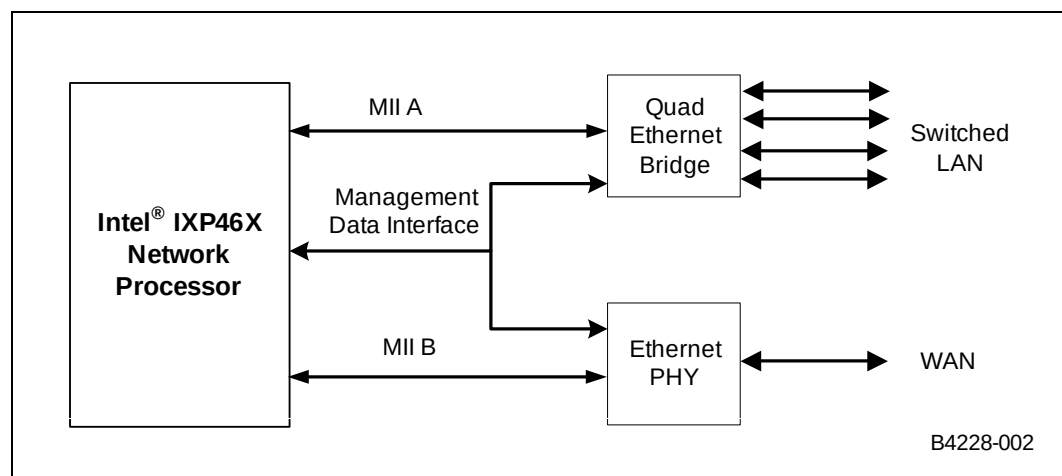
Note: Not all of the IXP45X/IXP46X network processors have this functionality. For details, see [Table 97, "Processors: Network Processor Functions" on page 222](#).

The IXP45X/IXP46X network processors contain three NPEs. All of the NPEs may be used to process Ethernet traffic utilizing the xMII (MII/SMII) interfaces. Each NPE core used for Ethernet traffic connects to a Transmit FIFO and Receive FIFO through the NPE Coprocessor interface. These Transmit and Receive FIFOs are used to buffer data between the Ethernet Media Access Controller (MAC) and the NPE core.

The Transmit FIFO, Receive FIFO, and MAC are contained in an NPE coprocessor unit called the Ethernet Coprocessor. The MAC contained in the Ethernet coprocessor is compliant to the IEEE 802.3 specification as well as handling flow control for the IEEE 802.3Q VLAN specification.

One Management Data Interface is shared between all of the xMII interfaces. The single Management Data Interface is used to configure the physical devices attached to each of the xMII interfaces. The Management Data Interface is accessible by manipulating the Management Data Registers associated with Ethernet MAC Coprocessor B0, see ["Register Descriptions Ethernet MACs" on page 239](#). [Figure 28](#) shows a typical application that may be used in connecting to the xMII interface. In addition to supporting an MII connection, the interface may be configured to support SMII interfaces.

Figure 28. Multiple Ethernet PHYS Connected to Processor

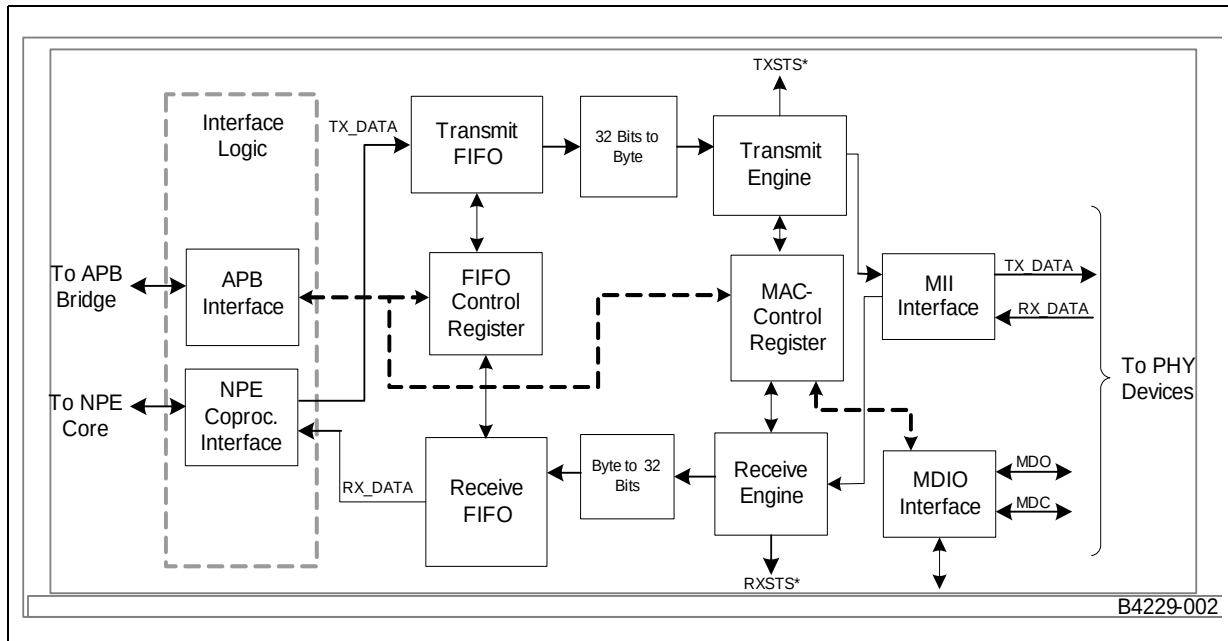


6.1 Ethernet Coprocessor

Each Ethernet coprocessor contains a Media Access Controller, a transmit data FIFO, and a receive data FIFO.

Figure 29 displays a block diagram of a single Ethernet coprocessor.

Figure 29. Ethernet Coprocessor Interface



The Ethernet Coprocessor communicates to peripheral devices and the remainder of the IXP45X/IXP46X network processors over four interfaces:

- APB Interface
- NPE Coprocessor Interface
- MII/SMII Interface
- Management Data Interface

6.1.1 Ethernet Coprocessor APB Interface

The APB interface is used to allow the Intel XScale processor to communicate directly to configuration and control registers utilized by the Media Access Controller. The Ethernet coprocessor's APB interface will be used to configure the Ethernet MAC, monitor Ethernet status, and configure the physical devices connected via the MII interfaces. The physical devices connected to the MII interface will be configured using the shared Management Data Interface.

6.1.2 Ethernet Coprocessor NPE Interface

The NPE Coprocessor Interface is used to communicate between the Ethernet coprocessor and the NPE core. The NPE coprocessor interface will be used to transfer incoming and outgoing data traffic to and from the NPE core. The NPE core — along with other coprocessor — will take the Ethernet data and perform data manipulation, forward the data to the SDRAM, and update the Queue Manager.



6.1.3 Ethernet Coprocessor MDIO Interface

The Management Data Interface is a two-wire interface that supports configuration of an xMII Interface on the IXP45X/IXP46X network processors. The Management Data Interface consists of the Management Data Input/Output (MDIO) signal and the Management Data Clock (MDC). The Management Data Input/Output signal is a bidirectional signal that is used to transfer control, configuration, and status information between the IXP45X/IXP46X network processors and any peripheral devices connected to the xMII interfaces.

The Ethernet Coprocessor is initiated three times to support three Ethernet PHYs outside the device. However, only the MDC and MDIO pins of one of the coprocessors (Ethernet B) were brought out and they are used to program both Ethernet PHYs.

The Management Data Clock can be configured as an input or an output, enabling the IXP45X/IXP46X network processors to source the Management Data Clock or enable an external device to source the clock. The Management Data Clock is used to clock the data sent on the Management Data Input/Output Signal.

Data transfers will be initiated over the MDIO using the MDIO Command Register (MDIOCMD). The MDIO Command Register is broken into four 8-bit registers that make up a full 32-bit command word.

If data is to be sent to the PHY over the MDIO interface, the Intel XScale processor will write a value to each of the four command words in sequential order:

- MDIO Command 1 (MDIOCMD1) Register and MDIO Command 2 (MDIOCMD2) Register will contain the 16 bits of data that the destination PHY will receive.
- MDIO Command 3 (MDIOCMD3) Register and MDIO Command 4 (MDIOCMD4) Register will determine which PHY number is to be addressed, the internal register of the addressed PHY, the direction of the access (read/write), and when to begin the access.

There can be a limit of 32 physical ports with a limit of 32 registers per physical port that may be addressed.

MDIOCMD3 makes up bits (23:16) of MDIOCMD and MDIOCMD4 make up bits (31:24):

- Bits (25:21) of MDIOCMD are used to select the physical interface that is to accept the transmitted data or return the requested data.
- Bits (20:16) of MDIOCMD are used to select the register within the physical interface that is to accept the transmitted data or return the requested data.
- Bit 26 of MDIOCMD is used to determine if the requested command is a read or a write:
 - Writing logic 1 to this bit will cause the transaction to be a write.
 - Writing logic 0 to this bit will cause the transaction to be a read.

Setting Bit 31 of the MDIO Command (MDIOCMD) Register to logic 1 will initiate the transfer. Bit 31 of MDIOCMD will remain at logic 1 until the transaction is complete.

Figure 30 shows an example of the data being written from the xMII Management Master (IXP45X/IXP46X network processors) to a physical interface (PHY) using the MDIO interface.

As stated previously, when bit 26 of the MDIO Command (MDIOCMD) Register is set to logic 0 the Intel XScale processor is requesting a read from a physical interface device using the MDIO interface. The data that the physical interface returns from the MDIO signal will be captured in the MDIO STATUS (MDIOSTS) Register.



The MDIO Status Register is broken into four 8-bit registers. The data returned from the physical interface will be captured in MDIO Status 0 (MDIOSTS0) Register and MDIO Status 1 (MDIOSTS1) Register:

- MDIO Status 0 (MDIOSTS0) Register corresponds to bits (7:0) of the MDIO Status (MDIOSTS) Register.
- MDIO Status 1 (MDIOSTS1) Register corresponds to bits (15:8) of the MDIO Status (MDIOSTS) Register.
- Bits (30:16) of the MDIO Status (MDIOSTS) Register are reserved and will return zeros when read.
- Bit 31 of the MDIO Status (MDIOSTS) Register will indicate the condition of the read.
 - If logic 1 is read from bit 31 of the MDIO Status (MDIOSTS) Register after a read transaction from the physical interface is complete, the read contained an error and should be disregarded.
 - If logic 0 is read from bit 31 of the MDIO Status (MDIOSTS) Register after a read transaction from the physical interface is complete, the read was valid and error free.

Figure 31 shows an example of the data being read from the physical interface (PHY) by the xMII Management Master (IXP45X/IXP46X network processors) using the MDIO interface. These registers should be manipulated using Intel-supplied APIs.

Figure 30. MDIO Write

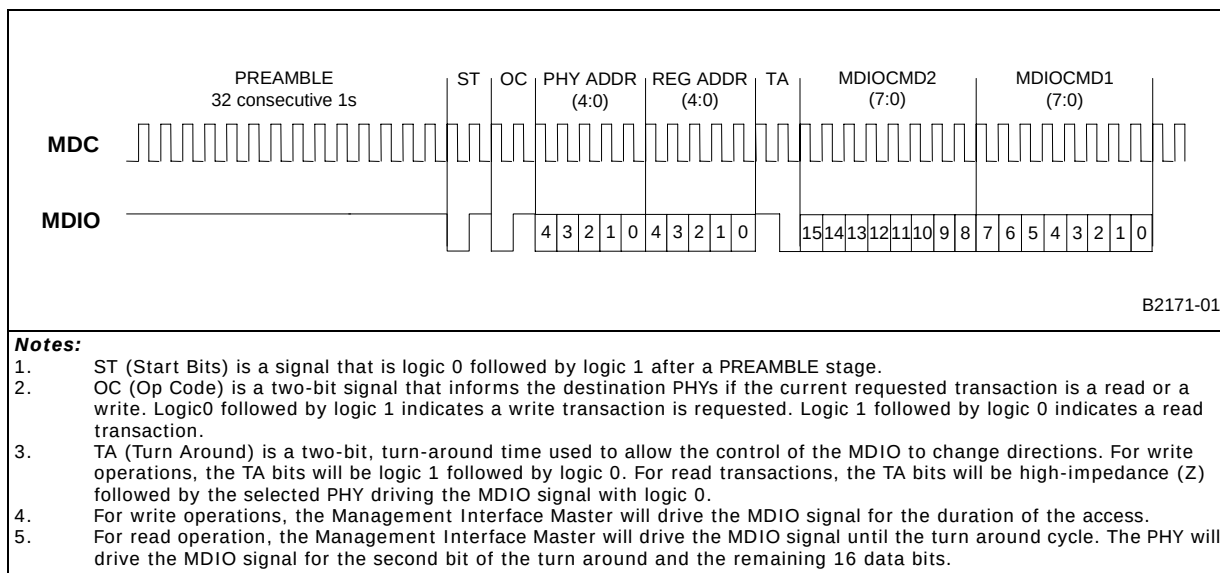
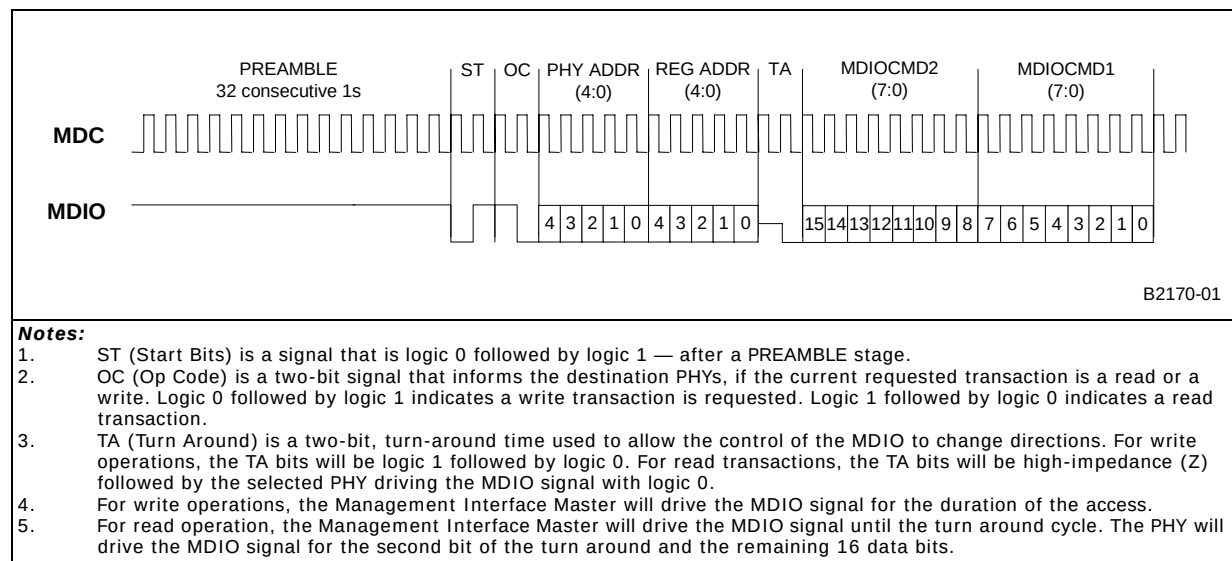


**Figure 31. MDIO Read**

6.1.4 Transmitting Ethernet Frames with MII Interfaces

Using processor API calls, the Intel XScale processor can request that packets be transmitted by the xMII Interface. The Intel XScale processor prepares an Ethernet packet to be transmitted. When the preparation is complete, the Intel XScale processor uses the Intel-supplied API calls to inform the Ethernet NPE that a packet is ready to be transmitted. The Ethernet NPE fetches the packet from the memory attached to the IXP45X/IXP46X network processors and forwards the data over the NPE coprocessor interface to the 256-byte Transmit FIFO contained in the Ethernet coprocessor.

Once the data has reached a predefined trigger point — known as the Buffer Size for Transmit Register (TXBUFFSIZE) in the Transmit FIFO — or the End-of-Frame signal is received, a data packet will begin to be transmitted over the MII interface. For each frame transmitted, TXBUFFSIZE holds the minimum number of bytes that must be contained in the transmit FIFO before the frame transmission may start. If the total size of the frame is less than the Buffer Size for Transmit Register, the frame will be transmitted when the end-of-frame signal is received.

When entries leave the bottom of the Transmit FIFO, the entries are 32 bits. The settings of the Buffer Size for Transmit Register (TXBUFFSIZE), the Threshold for Partially Full (THRESHPF), and the Threshold for Partially Empty (THRESHPE) are tightly coupled with the code written for the NPE core. Manipulation of the values will result in unpredictable behavior.

- The Threshold for Partially Full parameter sets a status flag going to the NPE core when the number of entries in the Transmit or Receive FIFOs is larger than the value programmed in this register.
- The Threshold for Partially Empty parameter sets a status flag going to the NPE core when the number of entries in the Transmit or Receive FIFOs is smaller than the value programmed in this register.

After the data begins leaving the FIFO, the data is sent through a converter function that is used to convert the bits from 32-bits to byte-wide entries to supply to the Transmit Engine. The Transmit Engine will take the bytes supplied from the converter,



manipulate the data as defined by MAC control registers and forward the data over the xMII interface as 4-bit nibbles in the case of MII operation and a serial stream for SMII modes of operation.

The Transmit Engine can be configured using processor API calls to:

- Append a Frame Check Sequence to the end of a transmitted frame
- Autonomously append bytes to frames that are smaller than the minimum frame size (64 bytes)
- Enable/Disable transmit retries
- Set the number of times a frame can be retried due to collision conditions before being dropped
- Select half- or full-duplex mode of operation
- Select a one- or two-part deferral to be used
- Enable/Disable the Transmit Engine

A Frame Check Sequence (FCS) can be autonomously generated and appended to the end of each Ethernet frame. The Frame Check Sequence can be used to ensure proper delivery of data between two Ethernet devices. The Frame Check Sequence consists of a 4-byte Cyclic Redundancy Generator that adheres to the polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The Frame Check Sequence will be computed over all fields beginning after the Start-of-Frame Delimiter (SFD) and up to the Frame Check Sequence (FCS) value. Autonomous insertion of the Frame Check Sequence into the transmitted frame can be enabled/disabled by setting bit 4 of Transmit Control Register 1 (TXCTRL1):

- Setting this bit to logic 1 will cause a CRC value to be generated and inserted into the Frame Check Sequence field of the transmitted frame.
- Setting this bit to logic 0 will cause a transmitted frame to be sent without a Frame Check Sequence attached.

Transmit Control Register 1 can be accessed directly, but Intel recommends that the Transmit Control Register 1 values be manipulated through Intel-supplied APIs. Failure to use the Intel-supplied APIs will result in unpredictable results.

The Transmit Engine can also be configured to append additional bytes to frames that are smaller than the 64-byte frame minimum. When the transmit engine observes the length field is smaller than 64 bytes – and bit 3 of Transmit Control Register 1 is set to logic 1 – the transmit engine will append additional bytes to equal the 64-byte minimum size. A CRC value will be calculated over the appended bytes; the appended bytes will be all zeros.

The IXP45X/IXP46X network processors also provide the capability to enable or disable transmit retries. When bit 2 of the Transmit Control Register 1 is set to logic 1 and collisions occur, the transmit engine will attempt to retry sending the packet up to the maximum number of transmit retries specified in bits (3:0) of Transmit Control Register 2 (TXCTRL2). A maximum of 16 retries can be attempted.

Note:

Setting bit 1 of TXCTRL1 to logic 1 configures a device in half-duplex mode of operation. Setting the bit to logic 0 configures the device to full-duplex mode.

- If a packet is being transmitted and a collision is detected *prior* to 64 bytes (minimum packet size) of the packet are transmitted, the transmit engine will rewind the transmit FIFO pointer to the beginning of the frame that is being



transmitted and attempt to send the frame after a calculated back-off time, based upon the Slot Time (SLOTTIME) Register.

- If a packet is being transmitted and a collision is detected *after* 64 bytes (minimum packet size) of a packet are transmitted, the transmit engine will forward the transmit FIFO pointer to the beginning of the next packet that is to be transmitted and discard the current packet that is being transmitted.

In a properly-configured network, a collision should not occur after the first 64 bytes of a packet are sent. The back-off time algorithm adheres to the IEEE 802.3 specifications "truncated binary exponential algorithm".

If the xMII interface is configured in half-duplex mode of operation, the transmit interface must listen to the activity on the line for a defined wait period, called a transmit deferral period. If there is any transmit or receive activity on the medium, the transmit interface will defer the transmission of the packet that the transmit engine has been requested to send.

When the transmit engine detects the medium has gone silent, the transmit engine will continue to defer for a period of time equal to the inter-frame spacing. The deferral period will be assigned by the transmit deferral parameters. The transmit deferral time can be specified as a one-part, or optional two-part, deferral by manipulating bit 5 of Transmit Control Register 1:

- Setting this bit to logic 0 enables the one-part transmit deferral.
- Setting this bit to logic 1 enables the optional, two-part transmit deferral.

If the transmit engine has a frame ready to transmit and one-part transmit deferral is selected, the transmit engine will wait for the medium to go silent and then wait for the time period specified in the Transmit Deferral Register (TXDEFPARS). The deferral period will be the number of transmit clock cycles specified by the 8-bit Transmit Deferral Register minus three transmit clock cycles. The Single Transmit Deferral parameter specifies the Inter Frame Gap.

In the two-part deferral process, the deferral period is defined using the Transmit Two Part Deferral Parameters 1 Register (TX2PARTDEFPARS1) and Transmit Two Part Deferral Parameters 2 Register (TX2PARTDEFPARS2). The values specified in these two registers when added together should not be less than the minimum Inter Frame Gap to ensure fair access to the medium.

Transmit Two Part Deferral Parameters 1 Register is typically set to two-thirds of the Inter Frame Gap and Transmit Two Part Deferral Parameters 2 Register is typically set to the remaining one-third.

If the transmit engine has a frame ready to transmit and two-part transmit deferral is selected, the transmit engine will wait for the medium to go silent and then wait for the time period specified in the Transmit Two Part Deferral Parameters 1 Register:

- If the medium is not silent during this first part of the deferral, the deferral counter is reset.
- If the medium is silent during the first part of the deferral, transmit engine continues to wait for the time period specified in the Transmit Two Part Deferral Parameters 2 Register.

When the xMII interface of the IXP45X/IXP46X network processors is configured in full-duplex mode of operation, the two-part transmit deferral parameters and the back-off times will not be utilized for data transmission.

Refer to IEEE 802.3 Section 4.2.3.2.1 for more information on deference.



The transmit deferral registers and the Transmit Control Register can be accessed directly, but Intel recommends that these register values be manipulated through Intel-supplied APIs. Failure to use the Intel-supplied APIs will result in unpredictable results.

In order to operate the interface in SMII mode, the Ethernet registers will be configured as if they are in MII mode of operation. Any setting produced in MII mode should translate to the same mode of operation when the interface is configured to SMII mode.

The value of TXCTRL1 can be accessed directly, but Intel recommends that TXCTRL1 values be manipulated through Intel-supplied APIs. Failure to use the Intel-supplied APIs will result in unpredictable results.

6.1.5 Receiving Ethernet Frames with MII Interfaces

When data is received using the xMII interface, the Receive engine will be used to convert the data from:

- 4-bit nibbles into 8-bit bytes when configured in MII mode of operation.
- 1-bit serial stream into 8-bit bytes when configured in SMII mode of operation.

The receive interface will then send the data to a byte-to-32-bit-word converter. The 32-bit word will be written into the 256-byte receive FIFO.

A flag will be generated to inform the NPE that the receive FIFO has new data to be removed after the Threshold for Partial Full/Empty value has been reached. The Receive Engine implements the following functions:

- Enable/Disable the receive engine
- Implements uni-cast/multi-case/broadcast, single-address filtering
- Checks for runt frames
- Checks for valid length/type fields
- Removes padded bits added to frames
- Implements the Frame-Check Sequence Algorithm

When new receive data is detected, the Receive Engine will look to see if the address filtering checks are enabled. If the address filtering checks are enable by setting bit 5 of the Receive Control Register 1 (RXCTRL1) to logic 1, the Receive Engine will obtain the destination address from the received frame, check the destination address against the address filtering parameters, and pass the frame to the 8-bit to 32-bit framing engine, if the test passed.

The Receive Engine is capable of filtering broadcast frames, multi-cast frames, and uni-cast frames. The frame filtering for multi-cast frames and uni-cast frames is controlled by registers called the Address Mask Register (ADDRMASK), the Address Register (ADDR), and the Uni-Cast Register (UNIADDR).

Determining if the 48-bit destination address of the received frame contains all logic 1s, filters the broadcast frames. Setting bit 7 of Receive Control Register 1 (RXCTRL1) to logic 1 prevents received broadcast frames from being sent to the NPE. (For more information see, [“Receive Control 1” on page 250.](#))

The multi-cast frame filtering and uni-cast frame filtering can be enabled/disabled by setting bit 5 of Receive Control Register 1 (RXCTRL1). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all non-broadcast frames to be sent to the NPE. The NPE can then operate in promiscuous mode and implement a more comprehensive filtering algorithm if required.



Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 1 enables filtering for uni-cast frames that are received. When Uni-Cast frames are detected, the destination address of the received frame must match exactly the value contained in the Uni-Cast Address Register (UNIADDR). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all uni-cast frames to be passed to the NPE.

For example, if the Uni-Cast Address Register — which is broken into six bytes — contains the value of 0x00ABCDEF1234 and uni-cast address filtering is enabled: Any uni-cast frames that are received (determined by bit 47 being set to logic 0) — with anything other than 0x00ABCDEF1234 — will be discarded.

Multi-cast is when frames are sent to multiple destinations from a single source with a single-frame transmission. The Address Mask Register (ADDRMASK) and the Address Register (ADDR) Multi-Cast frames can be used to filter multi-cast frames or frames with common addresses. The address mask is used to tell what each destination address has in common.

For example, bytes 3 and bytes 4 of every address in this group contain the same value. The Address Mask Register (ADDRMASK) would contain a hexadecimal value of 0x00FFFF000000. The Address Mask Register will be a logical “AND” with the Address Register and then a logical “AND” with the destination address. The result of the logical “AND” values will be compared to see if they match. If they match, the frame will be passed to the NPE via the remaining logic.

The values seen were as follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D2-47-63-21

Notice that the underlined values are all that matters for the comparison because of the address mask. The frame in this example would be forwarded to the next phase of the receive logic. An example of a frame that would be dropped due to address filtering follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D3-47-63-21

To disable the multi-cast address filtering feature set the Address Mask Register to all logic 0s.

After the received frame has passed all address filtering checks, the Receive Engine will capture the length/type field. If the length/type is determined to be a length value field and the length is less than 64 bytes, the Receive Engine will remove any padded bytes (assuming bit 1 of Receive Control Register 1 is set to logic 1) and capture the remaining data. Padded bytes will not be removed from the received packet when bit 1 of Receive Control Register is set to logic 0.

If the packet is less than 64 bytes and there are no padded bytes, the packet is determined to be a runt frame. The Receive Engine has the capability to discard these frames or forward the frames to the NPE via the remaining receive logic.

Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 1 informs the Receive Engine to allow the packet to be sent to the NPE. Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 0 informs the Receive Engine to terminate the reception of the runt packet and purge the runt packet from the rest of the receive logic.

If the length/type is determined to be a type field, the field is looked at for validity. Invalid frames are purged from the receive logic.



Once the received frame has passed the frame validity checks, the received frame will be checked for integrity using the Frame-Check Sequence Algorithm called out in the transmit-frame section. If the frame passes the Frame Check Sequence, the frame will be forwarded on to the NPE via the remaining receive logic. If the frame fails the Frame Check Sequence, the frame will be discarded along with purging all of the remaining receive logic of the frames contents.

Padded bytes will be included in the calculation of the Receive Engine's Frame Check Sequence for frames that were transmitted and were smaller than the 64-byte minimum frame size. A status flag will be sent to the NPE to inform the NPE what to do with the received frame.

In addition to the above features, the xMII receive interface allows some features to be used for test and debug. The transmit interface can be looped back to the receive interface by setting bit 4 of Receive Control Register 1 (RXCTRL1) to logic 1. In loop-back mode, the Receive Engine will receive all of the data sent by the Transmit Engine.

The loop-back feature allows software developers to develop their application code and test the code before they start dealing with physical interface problems. Setting bit 0 of Receive Control Register 1 (RXCTRL1) to logic 1 enables the Receive Engine. This feature allows all initialization of the product to occur prior to bring the receive interface online. Bit 0 of Receive Control Register 2 (RXCTRL2) enables deferral checking on the receive side. The IXP45X/IXP46X network processors do not use the receive side deferral checking feature.

Bit 0 of Receive Control Register 2 (RXCTRL2) must be set to logic 0 for proper operation. Failure to do so will result in unpredictable behavior.

Intel recommends that the register values described in this section be manipulated through Intel-supplied APIs. Failure to use the Intel supplied APIs will result in unpredictable results.

In order to operate the interface in SMII mode of operation, the ethernet registers will be configured as if they are in MII mode of operation. Any setting produced in MII mode of operation should translate to the same mode of operation when the interface is configured to SMII mode of operation.

6.1.6 General Ethernet Coprocessor Configuration

The Ethernet coprocessor contains various other registers that are used to configure the interface. Some of these registers are included due to the generic nature of the Ethernet coprocessor. Other registers are added to allow greater flexibility in configuration of the Ethernet coprocessor.

The Threshold for Internal Clock (THRESH_INTCLK) Register is used to determine the frequency relationship between the xMII interface and the host processor that is used to control the xMII interface. The value in the Threshold for Internal Clock (THRESH_INTCLK) Register will be manipulated based upon the ratio of $\text{PHY\_CLK\_SPEED/HOST\_CLK\_SPEED}$.

The physical interface clock speed will be divided by the host-side clock speed and then rounded to the nearest whole number. The value from this calculation will be written to the Threshold for Internal Clock (THRESH_INTCLK) Register. For the IXP45X/IXP46X network processors, the value contained in the Threshold for Internal Clock (THRESH_INTCLK) Register must always be set to hexadecimal 0x01. Failure to do so will result in unpredictable behavior. The value of the Threshold for Internal Clock (THRESH_INTCLK) Register will be pre-programmed by the Intel-supplied APIs to the proper value. Manipulation of this value will result in unpredictable behavior.



The Core Control (CORE_CONTROL) Register was added to allow the Intel XScale processor to have some control over the Ethernet coprocessor. Configuring bit 4 of the Core Control (CORE_CONTROL) Register can configure as an input or as an output the MDC clock.

Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP45X/IXP46X network processors to drive the MDC clock. Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 0 enables an external source to drive the MDC clock. The IXP45X/IXP46X network processors become a recipient of that MDC clock.

Configuring bit 3 of the Core Control (CORE_CONTROL) Register can configure the Transmit Engine to send a JAM sequence if a new packet starts to be received. Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP45X/IXP46X network processors to send a JAM sequence if a new packet is received.

Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 0 allows the Transmit Engine to function in normal mode of operation. Configuring bit 2 of the Core Control (CORE_CONTROL) Register causes the Transmit FIFO to be flushed. Any packets that are currently in the Transmit FIFO are discarded.

Setting bit 2 of the Core Control (CORE_CONTROL) Register to logic 1 clears the xMII Interface Transmit FIFO. Setting bit 2 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Transmit FIFO to resume normal mode of operation.

Configuring bit 1 of the Core Control (CORE_CONTROL) Register causes the Receive FIFO to be flushed. Any packets that are currently in the Receive FIFO are discarded. Setting bit 1 of the Core Control (CORE_CONTROL) Register to logic 1 clears the xMII Interface Receive FIFO. Setting bit 1 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Receive FIFO to resume normal mode of operation. Bit 0 of the Core Control (CORE_CONTROL) Register controls the reset state of the Media Access Controller (MAC) contained within the Ethernet coprocessor.

Setting bit 0 of the Core Control (CORE_CONTROL) Register back to logic 1 causes the Media Access Controller to be reset. De-assertion (setting bit 0 to logic 0) allows the Media Access Controller to resume operation in a fully reset state.

This register needs to be manipulated using Intel-supplied APIs. Failure to manipulate this register with Intel-supplied APIs will result in unpredictable behavior.

The Random-Seed Register is an 8-bit register used to support PHYs that support Auto MDI/MDI-X detection. The Random-Seed Register value is the value that is used to feed the Linear-Feedback Shift Register that is used to select which configuration to start initialization.

The Random-Seed Register needs to be manipulated using Intel-supplied APIs. Failure to manipulate this could result in unpredictable behavior.

6.2 Register Descriptions Ethernet MACs

The internal registers shown below are accessible via the APB bus interface. Unspecified addresses are reserved and should not be written; if read, a zero value will be returned.



Table 100. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

All of the Ethernet internal configuration and control registers are directly readable and writable by the Intel XScale processor via APB bus. All registers for all MII interfaces will match the detailed register descriptions contained in this chapter.

6.2.1 Ethernet MAC 0 on NPE B

Table 101. Ethernet MAC 0 on NPE B (Sheet 1 of 2)

Address	Description
0xC800 9000	Transmit Control 1
0xC800 9004	Transmit Control 2
0xC800 9010	Receive Control 1
0xC800 9014	Receive Control 2
0xC800 9020	Random Seed
0xC800 9030	Threshold For Partial Empty
0xC800 9038	Threshold For Partial Full
0xC800 9040	Buffer Size For Transmit
0xC800 9050	Transmit Single Deferral Parameters
0xC800 9054	Receive Deferral Parameters
0xC800 9060	Transmit Two Part Deferral Parameters 1
0xC800 9064	Transmit Two Part Deferral Parameters 2
0xC800 9070	Slot Time
0xC800 9080	MDIO Command 1
0xC800 9084	MDIO Command 2
0xC800 9088	MDIO Command 3
0xC800 908C	MDIO Command 4
0xC800 9090	MDIO Status 1
0xC800 9094	MDIO Status 2
0xC800 9098	MDIO Status 3
0xC800 909C	MDIO Status 4
0xC800 90A0	Address Mask 1
0xC800 90A4	Address Mask 2
0xC800 90A8	Address Mask 3
0xC800 90AC	Address Mask 4
0xC800 90B0	Address Mask 5

**Table 101. Ethernet MAC 0 on NPE B (Sheet 2 of 2)**

Address	Description
0xC800 90B4	Address Mask 6
0xC800 90C0	Address 1
0xC800 90C4	Address 2
0xC800 90C8	Address 3
0xC800 90CC	Address 4
0xC800 90D0	Address 5
0xC800 90D4	Address 6
0xC800 90E0	Threshold For Internal Clock
0xC800 90F0	Unicast Address 1
0xC800 90F4	Unicast Address 2
0xC800 90F8	Unicast Address 3
0xC800 90FC	Unicast Address 4
0xC800 9100	Unicast Address 5
0xC800 9104	Unicast Address 6
0xC800 91FC	Core Control

6.2.2 Ethernet MAC 1 on NPE B

Table 102. Ethernet MAC 1 on NPE B (Sheet 1 of 2)

Address	Description
0xC800 D000	Transmit Control 1
0xC800 D004	Transmit Control 2
0xC800 D010	Receive Control 1
0xC800 D014	Receive Control 2
0xC800 D020	Random Seed
0xC800 D030	Threshold For Partial Empty
0xC800 D038	Threshold For Partial Full
0xC800 D040	Buffer Size For Transmit
0xC800 D050	Transmit Single Deferral Parameters
0xC800 D054	Receive Deferral Parameters
0xC800 D060	Transmit Two Part Deferral Parameters 1
0xC800 D064	Transmit Two Part Deferral Parameters 2
0xC800 D070	Slot Time
0xC800 D080	MDIO Command 1
0xC800 D084	MDIO Command 2



Table 102. Ethernet MAC 1 on NPE B (Sheet 2 of 2)

Address	Description
0xC800 D088	MDIO Command 3
0xC800 D08C	MDIO Command 4
0xC800 D090	MDIO Status 1
0xC800 D094	MDIO Status 2
0xC800 D098	MDIO Status 3
0xC800 D09C	MDIO Status 4
0xC800 D0A0	Address Mask 1
0xC800 D0A4	Address Mask 2
0xC800 D0A8	Address Mask 3
0xC800 D0AC	Address Mask 4
0xC800 D0B0	Address Mask 5
0xC800 D0B4	Address Mask 6
0xC800 D0C0	Address 1
0xC800 D0C4	Address 2
0xC800 D0C8	Address 3
0xC800 D0CC	Address 4
0xC800 D0D0	Address 5
0xC800 D0D4	Address 6
0xC800 D0E0	Threshold For Internal Clock
0xC800 D0F0	Unicast Address 1
0xC800 D0F4	Unicast Address 2
0xC800 D0F8	Unicast Address 3
0xC800 D0FC	Unicast Address 4
0xC800 D100	Unicast Address 5
0xC800 D104	Unicast Address 6
0xC800 D1FC	Core Control



6.2.3 Ethernet MAC 2 on NPE B

Table 103. Ethernet MAC 2 on NPE B (Sheet 1 of 2)

Address	Description
0xC800 E000	Transmit Control 1
0xC800 E004	Transmit Control 2
0xC800 E010	Receive Control 1
0xC800 E014	Receive Control 2
0xC800 E020	Random Seed
0xC800 E030	Threshold For Partial Empty
0xC800 E038	Threshold For Partial Full
0xC800 E040	Buffer Size For Transmit
0xC800 E050	Transmit Single Deferral Parameters
0xC800 E054	Receive Deferral Parameters
0xC800 E060	Transmit Two Part Deferral Parameters 1
0xC800 E064	Transmit Two Part Deferral Parameters 2
0xC800 E070	Slot Time
0xC800 E080	MDIO Command 1
0xC800 E084	MDIO Command 2
0xC800 E088	MDIO Command 3
0xC800 E08C	MDIO Command 4
0xC800 E090	MDIO Status 1
0xC800 E094	MDIO Status 2
0xC800 E098	MDIO Status 3
0xC800 E09C	MDIO Status 4
0xC800 E0A0	Address Mask 1
0xC800 E0A4	Address Mask 2
0xC800 E0A8	Address Mask 3
0xC800 E0AC	Address Mask 4
0xC800 E0B0	Address Mask 5
0xC800 E0B4	Address Mask 6
0xC800 E0C0	Address 1
0xC800 E0C4	Address 2
0xC800 E0C8	Address 3



Table 103. Ethernet MAC 2 on NPE B (Sheet 2 of 2)

Address	Description
0xC800 E0CC	Address 4
0xC800 E0D0	Address 5
0xC800 E0D4	Address 6
0xC800 E0E0	Threshold For Internal Clock
0xC800 E0F0	Unicast Address 1
0xC800 E0F4	Unicast Address 2
0xC800 E0F8	Unicast Address 3
0xC800 E0FC	Unicast Address 4
0xC800 E100	Unicast Address 5
0xC800 E104	Unicast Address 6
0xC800 E1FC	Core Control

6.2.4 Ethernet MAC 2 on NPE B

Table 104. Ethernet MAC 2 on NPE B (Sheet 1 of 2)

Address	Description
0xC800 F000	Transmit Control 1
0xC800 F004	Transmit Control 2
0xC800 F010	Receive Control 1
0xC800 F014	Receive Control 2
0xC800 F020	Random Seed
0xC800 F030	Threshold For Partial Empty
0xC800 F038	Threshold For Partial Full
0xC800 F040	Buffer Size For Transmit
0xC800 F050	Transmit Single Deferral Parameters
0xC800 F054	Receive Deferral Parameters
0xC800 F060	Transmit Two Part Deferral Parameters 1
0xC800 F064	Transmit Two Part Deferral Parameters 2
0xC800 F070	Slot Time
0xC800 F080	MDIO Command 1
0xC800 F084	MDIO Command 2
0xC800 F088	MDIO Command 3

**Table 104. Ethernet MAC 2 on NPE B (Sheet 2 of 2)**

Address	Description
0xC800 F08C	MDIO Command 4
0xC800 F090	MDIO Status 1
0xC800 F094	MDIO Status 2
0xC800 F098	MDIO Status 3
0xC800 F09C	MDIO Status 4
0xC800 F0A0	Address Mask 1
0xC800 F0A4	Address Mask 2
0xC800 F0A8	Address Mask 3
0xC800 F0AC	Address Mask 4
0xC800 F0B0	Address Mask 5
0xC800 F0B4	Address Mask 6
0xC800 F0C0	Address 1
0xC800 F0C4	Address 2
0xC800 F0C8	Address 3
0xC800 F0CC	Address 4
0xC800 F0D0	Address 5
0xC800 F0D4	Address 6
0xC800 F0E0	Threshold For Internal Clock
0xC800 F0F0	Unicast Address 1
0xC800 F0F4	Unicast Address 2
0xC800 F0F8	Unicast Address 3
0xC800 F0FC	Unicast Address 4
0xC800 F100	Unicast Address 5
0xC800 F104	Unicast Address 6
0xC800 F1FC	Core Control



6.2.5 Ethernet MAC on NPE A

Table 105. Ethernet MAC on NPE A (Sheet 1 of 2)

Address	Description
0xC800 C000	Transmit Control 1
0xC800 C004	Transmit Control 2
0xC800 C010	Receive Control 1
0xC800 C014	Receive Control 2
0xC800 C020	Random Seed
0xC800 C030	Threshold For Partial Empty
0xC800 C038	Threshold For Partial Full
0xC800 C040	Buffer Size For Transmit
0xC800 C050	Transmit Single Deferral Parameters
0xC800 C054	Receive Deferral Parameters
0xC800 C060	Transmit Two Part Deferral Parameters 1
0xC800 C064	Transmit Two Part Deferral Parameters 2
0xC800 C070	Slot Time
0xC800 C080	MDIO Command 1 [†]
0xC800 C084	MDIO Command 2 [†]
0xC800 C088	MDIO Command 3 [†]
0xC800 C08C	MDIO Command 4 [†]
0xC800 C090	MDIO Status 1 [†]
0xC800 C094	MDIO Status 2 [†]
0xC800 C098	MDIO Status 3 [†]
0xC800 C09C	MDIO Status 4 [†]
0xC800 C0A0	Address Mask 1
0xC800 C0A4	Address Mask 2
0xC800 C0A8	Address Mask 3
0xC800 C0AC	Address Mask 4
0xC800 C0B0	Address Mask 5
0xC800 C0B4	Address Mask 6
0xC800 C0C0	Address 1
0xC800 C0C4	Address 2
0xC800 C0C8	Address 3

**Table 105. Ethernet MAC on NPE A (Sheet 2 of 2)**

Address	Description
0xC800 C0CC	Address 4
0xC800 C0D0	Address 5
0xC800 C0D4	Address 6
0xC800 C0E0	Threshold For Internal Clock
0xC800 C0F0	Unicast Address 1
0xC800 C0F4	Unicast Address 2
0xC800 C0F8	Unicast Address 3
0xC800 C0FC	Unicast Address 4
0xC800 C100	Unicast Address 5
0xC800 C104	Unicast Address 6
0xC800 C1FC	Core Control
† The MDI interface on NPE-A is inactive. All external PHYs are configured via NPE-B.	

6.2.6 Ethernet MAC on NPE C

Table 106. Ethernet MAC on NPE C (Sheet 1 of 2)

Address	Description
0xC800 A000	Transmit Control 1
0xC800 A004	Transmit Control 2
0xC800 A010	Receive Control 1
0xC800 A014	Receive Control 2
0xC800 A020	Random Seed
0xC800 A030	Threshold For Partial Empty
0xC800 A038	Threshold For Partial Full
0xC800 A040	Buffer Size For Transmit
0xC800 A050	Transmit Single Deferral Parameters
0xC800 A054	Receive Deferral Parameters
0xC800 A060	Transmit Two Part Deferral Parameters 1
0xC800 A064	Transmit Two Part Deferral Parameters 2
0xC800 A070	Slot Time
0xC800 A080	MDIO Command 1 [†]
0xC800 A084	MDIO Command 2 [†]



Table 106. Ethernet MAC on NPE C (Sheet 2 of 2)

Address	Description
0xC800 A088	MDIO Command 3 [†]
0xC800 A08C	MDIO Command 4 [†]
0xC800 A090	MDIO Status 1 [†]
0xC800 A094	MDIO Status 2 [†]
0xC800 A098	MDIO Status 3 [†]
0xC800 A09C	MDIO Status 4 [†]
0xC800 A0A0	Address Mask 1
0xC800 A0A4	Address Mask 2
0xC800 A0A8	Address Mask 3
0xC800 A0AC	Address Mask 4
0xC800 A0B0	Address Mask 5
0xC800 A0B4	Address Mask 6
0xC800 A0C0	Address 1
0xC800 A0C4	Address 2
0xC800 A0C8	Address 3
0xC800 A0CC	Address 4
0xC800 A0D0	Address 5
0xC800 A0D4	Address 6
0xC800 A0E0	Threshold For Internal Clock
0xC800 A0F0	Unicast Address 1
0xC800 A0F4	Unicast Address 2
0xC800 A0F8	Unicast Address 3
0xC800 A0FC	Unicast Address 4
0xC800 A100	Unicast Address 5
0xC800 A104	Unicast Address 6
0xC800 C1FC	Core Control
[†] The MDI interface on NPE-C is inactive. All external PHYs are configured via NPE-B.	



6.2.7 Transmit Control 1

Register Name:		txctrl1																							
Hex Offset Address:		0xC8009000								Reset Hex Value:		0x00000000													
Register Description:		Transmit Control Register																							
Access: Read / Write.																									
31																		7	6	5	4	3	2	1	0
(Reserved)																			MII CFG	2PRT DEF	APP FCS	PAD EN	RET EN	HALFDUP	TX EN

Register		txctrl1
Bits	Name	Description
31:7		(Reserved)
6	MII config	0 = Configures the PHY interface as a MII.
5	Two-part deferral	1 = Causes the optional two part deferral to be used.
4	Append FCS	1 = Causes FCS to be computed and appended to transmit frames before they are sent to the PHY.
3	Pad enable	1 = Causes transmit frames less than to minimum frame size to be padded before they are sent to the PHY.
2	Retry enable	1 = Causes transmit frames to be retried until the maximum retry limit shown in the Transmit Control 2 Register is reached, when collisions occur.
1	Half duplex	1 = Half-duplex operation 0 = Full-duplex
0	Transmit enable	1 = Causes transmission to be enabled.

6.2.8 Transmit Control 2

Register Name:	txctrl2																											
Hex Offset Address:	0xC8009004					Reset Hex Value:				0x00000000																		
Register Description:	Transmit Control Register																											
Access: Read/Write.																												
31																								4	3			0
(Reserved)																							Maximum Retries					

Register		txctrl2
Bits	Name	Description
4:31	(Reserved)	
3:0	Maximum retries	Maximum number of retries for a packet when collisions occur.



6.2.9 Receive Control 1

Register Name:		rxctrl1																																		
Hex Offset Address:		0xC8009010								Reset Hex Value:		0x00000000																								
Register Description:		Receive Control Register																																		
Access: Read/Write.																																				
31																												8	7	6	5	4	3	2	1	0
(Reserved)																											BCDIS	RX RP	ADD FILT	LOOP EN	PSE EN	CRC	PAD STRP	RX EN		

Register		rxctrl1
Bits	Name	Description
31:8	(Reserved)	
7	Broadcast disable	Broadcast packets will only be dropped if broadcast disable is set to "1", the address mask register is NOT 00 00 00 00 00 00 and the address filter is enabled. Setting the address mask register to all 00 (don't care about the address) and setting the Broadcast disable bit to "1" (checking the address) at the same time is a contradiction and will result in broadcast packets still being received and the packet status (MCST_PKT / BCST_PKT) will be read back accordingly.
6	Receive runt packet	1 = Causes runt packets to be passed to the application logic. 0 = Runt packets are dropped.
5	Address filter enable	1 = Causes address filtering to take place. Non-broadcast packets are only passed to the application logic if they pass the address filter.
4	Loopback enable	1 = Causes loop-back operation. Note: In order for the loop-back operation to operate correctly, the Ethernet coprocessor requires synchronous and in-phase clocks to be provided on the rx_clk and tx_clk pins.
3	Pause enable	1 = Enables detection of Pause frames. Upon detecting a pause frame, data transmission is halted based on the data in the pause frame. The 2-byte data of the received pause frame indicates the time, as a number of 512 bit times, to halt the transmission.
2	Send CRC	1 = Causes the CRC data to be sent to the application logic.
1	Pad strip	1 = Causes the pad bytes to be stripped from receive data.
0	Receive enable	1 = Causes reception to be enabled.



6.2.10 Receive Control 2

Register Name:		rxctrl2																																	
Hex Offset Address:		0xC8009014																Reset Hex Value:		0x00000000															
Register Description:		Receive Control Register																																	
Access: Read/Write.																																			
31																													1	0					
(Reserved)																												RX DEF EN							

Register		rxctrl2
Bits	Name	Description
31:1	(Reserved)	
0	Receive deferral enable	1 = Enables receive deferral checking.

6.2.11 Random Seed

Register Name:	rndmseed																																					
Hex Offset Address:	0xC8009020								Reset Hex Value:	0x00000000																												
Register Description:	Random Seed Register																																					
Access: Read/Write.																																						
31																										8	7							0				
(Reserved)																													Random Seed									

Register		rndmseed
Bits	Name	Description
31:8	(Reserved)	
7:0	Random Seed	Random seed used for LFSR initialization in the back-off block.

6.2.12 Threshold For Partially Empty

Register Name:		threshpe																																			
Hex Offset Address:		0xC8009030								Reset Hex Value:		0x00000000																									
Register Description:		FIFO Partially Full/Empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.																																			
Access: Read/Write.																																					
31																												8	7								0
(Reserved)																								Partial Empty													



Register		threshpe
Bits	Name	Description
31:8	(Reserved)	
7:0	Partial Empty	Marks the partial empty thresholds of the Transmit FIFO and Receive FIFO. When the number of entries in the Transmit FIFO is less than or equal to the contents of this register, tx_fifo_p_empty is asserted. When the number of entries in the Receive FIFO is less than or equal to the contents of this register, rx_fifo_p_empty is asserted.

6.2.13 Threshold For Partially Full

Register Name:		threshpf																											
Hex Offset Address:		0xC8009038								Reset Hex Value:		0x00000000																	
Register Description:		FIFO Partially Full/Empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.																											
Access: Read/Write.																													
31																	8	7									0		
(Reserved)																				Partial Full									



6.2.15 Transmit Deferral Parameter

Register Name:		txdefpars																								
Hex Offset Address:		0xC8009050								Reset Hex Value:		0x00000000														
Register Description:		Transmit/Receive Deferral Parameters																								
Access: Read/Write.																										
31																8	7									0
(Reserved)																		Transmit Deferral								

Register		txdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Single deferral	Number of transmit clock cycles (tx_clk) in the transmit deferral period minus three, when single deferral is used for transmission (Transmit Control[15] = 0).

6.2.16 Receive Deferral Parameter

Register Name:		rxdefpars																								
Hex Offset Address:		0xC8009054								Reset Hex Value:		0x00000000														
Register Description:		Transmit/Receive Deferral Parameters																								
Access: Read/Write.																										
31																	8	7								0
(Reserved)																Receive Deferral										

Register		rxdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Receive Deferral	Number of receive clock cycles (rx_clk) in the receive deferral period minus three, when checking the Inter Frame Gap for packets received (Receive Control 2[0] = 0).

6.2.17 Transmit Two Part Deferral Parameters 1

Register Name:	tx2partdefpars1																									
Hex Offset Address:	0xC8009060								Reset Hex Value:	0x00000000																
Register Description:	Transmit Two Part Deferral Parameters Register.																									
Access: Read/Write.																										
31																		8	7							0
(Reserved)																			First Deferral Period							



Register		txdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	First deferral period	Number of transmit clock cycles (tx_clk) in the first deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

6.2.18 Transmit Two Part Deferral Parameters 2

Register Name:		tx2partdefpars2																																	
Hex Offset Address:		0xC8009064								Reset Hex Value:		0x00000000																							
Register Description:		Transmit Two Part Deferral Parameters Register																																	
Access: Read/Write.																																			
31																									8	7									0
(Reserved)																										Second Deferral Period									

Register		tx2defpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Second Deferral Period	Number of transmit clock cycles (tx_clk) in the second deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

6.2.19 Slot Time

Register Name:		slottime																																		
Hex Offset Address:		0xC8009070								Reset Hex Value:		0x00000000																								
Register Description:		Slot Time Register																																		
Access: Read/Write.																																				
31																										8	7									0
(Reserved)																								Slot Time												

Register		slottime
Bits	Name	Description
31:8	(Reserved)	
7:0	Slot Time	Slot time for back-off algorithm Expressed in number of tx_clk cycles. 128 in MII mode.

6.2.20 MDIO Commands Registers

Four registers make up the 32-bit MDIO Command:

- MDIO Command[31:24] — MDIO Command 4



- MDIO Command[23:16] — MDIO Command 3
- MDIO Command[15:8] — MDIO Command 2
- MDIO Command[7:0] — MDIO Command 1

The detailed bit descriptions follow the four commands' bit maps.

6.2.21 MDIO Command 1

Register Name:	mdiocmd1																										
Hex Offset Address:	0xC8009080								Reset Hex Value:				0x00000000														
Register Description:	MDIO Command 1 (8 Bits of 32-Bit Register).																										
Access: Read/Write.																											
																				7							0
																			MDIO Command [7:0]								

6.2.22 MDIO Command 2

Register Name:	mdiocmd2																										
Hex Offset Address:	0x C8009084								Reset Hex Value:				0x00000000														
Register Description:	MDIO Command Register																										
Access: Read/Write.																											
31																		8	7								0
(Reserved)																				MDIO_COMMAND [15:8]							

6.2.23 MDIO Command 3

Register Name:		mdiocm3																								
Hex Offset Address:		0x C8009088								Reset Hex Value:		0x00000000														
Register Description:		MDIO Command Register																								
Access: Read/Write.																										
31																8	7									0
(Reserved)																MDIO_COMMAND [23:16]										



6.2.24 MDIO Command 4

Register Name:		mdiocm4																														
Hex Offset Address:		0x C800908C								Reset Hex Value:		0x00000000																				
Register Description:		MDIO Command Register																														
Access: Read/Write.																																
31	30																							8	7							0
Go	(Reserved)																				MDIO_COMMAND[31:24]											

Register		MDIO Command
Bits	Name	Description
31	Go	Application logic sets this to 1 to start the MDIO access. This bit remains 1 during the access. When the access is finished, the Ethernet core resets this bit to 0.
30:27	(Reserved)	
26	MDIO Write	1 = MDIO write access 0 = MDIO read access.
25:21	PHY address	Physical address of the PHY to be accessed.
20:16	PHY Register	Register number of the PHY Register to be accessed.
15:0	Write data	Write data on MDIO write accesses.

6.2.25 MDIO Status Registers

Four registers make up the 32-bit MDIO status:

- MDIO Status[31:24] — MDIO Status 4
- MDIO Status[23:16] — MDIO Status 3
- MDIO Status[15:8] — MDIO Status 2
- MDIO Status[7:0] — MDIO Status 1

The detailed bit descriptions follow the four registers' bit maps.

6.2.26 MDIO Status 1

Register Name:		mdiosts1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
Hex Offset Address:		0x C8009090								Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
Register Description:		MDIO Status Register																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
Access: Read Only.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
31																												8	7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														</



6.2.27 MDIO Status 2

Register Name:		mdiosts2																									
Hex Offset Address:		0x C8009094								Reset Hex Value:		0x00000000															
Register Description:		MDIO Status Register																									
Access: Read Only.																											
31																		8	7								0
(Reserved)																			MDIO_STATUS[15:8]								

6.2.28 MDIO Status 3

Register Name:		mdiosts3																									
Hex Offset Address:		0x C8009098								Reset Hex Value:		0x00000000															
Register Description:		MDIO Status Register																									
Access: Read Only.																											
31																		8	7								0
(Reserved)																				MDIO_STATUS[23:16]							

6.2.29 MDIO Status 4

Register Name:		mdiosts4																								
Hex Offset Address:		0x C800909C								Reset Hex Value:		0x00000000														
Register Description:		MDIO Status Register																								
Access: Read Only.																										
31																		8	7							0
(Reserved)																			MDIO_STATUS[31:24]							

Register		MDIO Status
Bits	Name	Description
31	Successful read	0 = A successful read 1 = A read error. Read only.
30:16	(Reserved)	Read only.
15:0	Read data	Read only.

6.2.30 Address Mask Registers

Six registers make up the 48-bit Address Mask:

- Address Mask[47:40] — Address Mask 1
- Address Mask[39:32] — Address Mask 2
- Address Mask[31:24] — Address Mask 3



- Address Mask[23:16] — Address Mask 4
- Address Mask[15:8] — Address Mask 5
- Address Mask[7:0] — Address Mask 6

Example: Address Mask is 00-A0-24-D1-7F-02

- Address Mask 1 = 0x00
- Address Mask 2 = 0x00
- Address Mask 3 = 0x00
- Address Mask 4 = 0xFF
- Address Mask 5 = 0xFF
- Address Mask 6 = 0x00

The detailed bit descriptions follow the six registers' bit maps.

6.2.31 Address Mask 1

Register Name:		addrmask1																							
Hex Offset Address:		0x C80090A0								Reset Hex Value:		0x00000000													
Register Description:		Address Mask Register #1. First register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted																							
Access: Read/Write.																									
31																	8	7							0
(Reserved)																		ADDRESS MASK[7:0]							

6.2.32 Address Mask 2

Register Name:		addrmask2																									
Hex Offset Address:		0x C80090A4								Reset Hex Value:		0x00000000															
Register Description:		Address Mask Register #1. Second register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																									
Access: Read/Write.																											
31																	8	7									0
(Reserved)																		ADDRESS MASK[15:8]									



6.2.33 Address Mask 3

Register Name:		addrmask3																								
Hex Offset Address:		0x C80090A8								Reset Hex Value:		0x00000000														
Register Description:		Address Mask Register #1. Third register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																								
Access: Read/Write.																										
31																	8	7								0
(Reserved)																ADDRESS MASK [23:16]										

6.2.34 Address Mask 4

Register Name:		addrmask4																								
Hex Offset Address:		0x C80090AC								Reset Hex Value:		0x00000000														
Register Description:		Address Mask Register #1. Forth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																								
Access: Read/Write.																										
31																		8	7							0
(Reserved)																			ADDRESS MASK[31:24]							

6.2.35 Address Mask 5

Register Name:		addrmask5																																		
Hex Offset Address:		0x C80090B0								Reset Hex Value:		0x00000000																								
Register Description:		Address Mask Register #1. Fifth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																																		
Access: Read/Write.																																				
31																										8	7									0
(Reserved)																				ADDRESS MASK [39:32]																



6.2.36 Address Mask 6

Register Name:		addrmask6																	
Hex Offset Address:		0x C80090B4								Reset Hex Value:		0x00000000							
Register Description:		Address Mask Register #1. Sixth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																	
Access: Read/Write.																			
31											16	15					0		

Register		Address Mask
Bits	Name	Description
47:0	Address Mask1-6	Address Mask 1-6 is used with Address 1-6 for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.

6.2.37 Address Registers

Six registers make up the 48-bit Address:

- Address[47:40] — Address 1
- Address[39:32] — Address 2
- Address[31:24] — Address 3
- Address[23:16] — Address 4
- Address[15:8] — Address 5
- Address[7:0] — Address 6

Example: Address is 00-A0-24-D1-7F-02

- Address 1 = 0x00
- Address 2 = 0xA0
- Address 3 = 0x24
- Address 4 = 0xD1
- Address 5 = 0x7F
- Address 6 = 0x02

The detailed bit descriptions follow the six registers' bit maps.



6.2.38 Address 1

Register Name:		addr1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
Hex Offset Address:		0x C80090C0								Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Register Description:		Address Register #1. First register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
Access: Read/Write.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																										8	7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				</

6.2.39 Address 2

Register Name:		addr2																									
Hex Offset Address:		0x C80090C4								Reset Hex Value:		0x00000000															
Register Description:		Address Register #1. Second register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																									
Access: Read/Write.																											
31																		8	7								0
(Reserved)																				ADDRESS[15:8]							

6.2.40 Address 3

Register Name:		addr3																									
Hex Offset Address:		0x C80090C8								Reset Hex Value:		0x00000000															
Register Description:		Address Register #1. Third register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																									
Access: Read/Write.																											
31																		8	7								0
(Reserved)																				ADDRESS[23:16]							



6.2.41 Address 4

Register Name:		addr4																																					
Hex Offset Address:		0x C80090CC												Reset Hex Value:		0x00000000																							
Register Description:		Address Register #1. Forth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																																					
Access: Read/Write.																																							
31																													8	7									0
(Reserved)																								ADDRESS[31:24]															

6.2.42 Address 5

Register Name:		addr5																														
Hex Offset Address:		0x C80090D0								Reset Hex Value:		0x00000000																				
Register Description:		Address Register #1. Fifth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																														
Access: Read/Write.																																
31																								8	7							0
(Reserved)																							ADDRESS[39:32]									

6.2.43 Address 6

Register Name:		addr6																														
Hex Offset Address:		0x C80090D4								Reset Hex Value:		0x00000000																				
Register Description:		Address Register #1. Sixth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.																														
Access: Read/Write.																																
31																								8	7							0
(Reserved)																							ADDRESS[47:40]									

Register		Address
Bits	Name	Description
47:0	Address	Address Mask 1-6 is used with Address 1-6 for multi-cast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.



6.2.44 Threshold for Internal Clock

Register Name:	thresh_intclk																										
Hex Offset Address:	0xC80090E0								Reset Hex Value:	0x00000000																	
Register Description:	Threshold for Internal Clock Register																										
Access: Read/Write.																											
31																		8	7								0
(Reserved)																			CLOCK RATIO								

Register		thresh_intclk
Bits	Name	Description
31:8	(Reserved)	
7:0	Clock ratio	Holds ratio of PHY side clock (tx_clk or rx_clk) to sys_clk. PHY side clock frequency/application clock frequency. Round up for value to be written to this register. Example: 25 MHz PHY clock, 133 MHz application clock — Sets this register to 1. Any application, clock frequency greater than the tx_clk or rx_clk frequency will have a clock ratio of 1. Always set to 1 for the IXP45X/IXP46X network processors.

6.2.45 Unicast Address Registers

Six registers make up the 48-bit Unicast Address:

- Unicast Address[47:40] — Unicast Address 1
- Unicast Address[39:32] — Unicast Address 2
- Unicast Address[31:24] — Unicast Address 3
- Unicast Address[23:16] — Unicast Address 4
- Unicast Address[15:8] — Unicast Address 5
- Unicast Address[7:0] — Unicast Address 6

Example: Unicast Address is 00-A0-24-D1-7F-02

- Unicast Address 1 = 0x00
- Unicast Address 2 = 0xA0
- Unicast Address 3 = 0x24
- Unicast Address 4 = 0xD1
- Unicast Address 5 = 0x7F
- Unicast Address 6 = 0x02

The detailed bit descriptions follow the six registers' bit maps.



6.2.46 Unicast Address 1

Register Name:		uniaddr1																																		
Hex Offset Address:		0x C80090F0								Reset Hex Value:		0x00000000																								
Register Description:		Unicast Address Register #1. First register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.																																		
Access: Read/Write.																																				
31																										8	7									0
(Reserved)																									UNICAST ADDRESS[7:0]											

6.2.47 Unicast Address 2

Register Name:		uniaddr2																																		
Hex Offset Address:		0x C80090F4								Reset Hex Value:		0x00000000																								
Register Description:		Unicast Address Register #1. Second register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.																																		
Access: Read/Write.																																				
31																										8	7									0
(Reserved)																									UNICAST ADDRESS[15:8]											

6.2.48 Unicast Address 3

Register Name:		uniaddr3																																	
Hex Offset Address:		0x C80090F8								Reset Hex Value:		0x00000000																							
Register Description:		Unicast Address Register #1. Third register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.																																	
Access: Read/Write.																																			
31																								8	7									0	
(Reserved)																										UNICAST ADDRESS[23:16]									

6.2.49 Unicast Address 4

Register Name:		uniaddr4																																
Hex Offset Address:		0x C80090FC								Reset Hex Value:		0x00000000																						
Register Description:		Unicast Address Register #1. Forth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.																																
Access: Read/Write.																																		
31																								8	7									0
(Reserved)																						UNICAST ADDRESS[31:24]												



6.2.50 Unicast Address 5

Register Name:	uniaddr5																								
Hex Offset Address:	0x C8009100								Reset Hex Value:	0x00000000															
Register Description:	Unicast Address Register #1. Fifth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.																								
Access: Read/Write.																									
31																	8	7							0
(Reserved)																		UNICAST ADDRESS[39:32]							

6.2.51 Unicast Address 6

Register Name:	uniaddr6																									
Hex Offset Address:	0x C8009104								Reset Hex Value:	0x00000000																
Register Description:	Unicast Address Register #1. Sixth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE																									
Access: Read/Write.																										
31																		8	7							0
(Reserved)																			UNICAST ADDRESS[47:40]							

Register		Unicast Address
Bits	Name	Description
47:0		Unicast Address 1-6 are matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.

6.2.52 Core Control

Register Name:		core_control																												
Hex Offset Address:		0xC80091FC								Reset Hex Value:		0x00000000																		
Register Description:		Controls key functions of the core																												
Access: Read/Write.																														
31																									5	4	3	2	1	0
(Reserved)																									mdc_en	send_jam	clr_tx_err	clr_rx_err	rst_mac	



Register		core_control
Bits	Name	Description
31:5	(Reserved)	
4	Mdc_en	1 = Configures the MDC as an output clock. Set to 1 for the IXP45X/IXP46X network processors MAC0 on NPE B. This bit is set as <i>Reserved</i> for MACs on NPE A and NPE C
3	Send_jam	1 = Causes a jam sequence to be sent if reception of a packet begins.
2	clr_tx_err	Assertion ("1") causes the Transmit FIFO to be flushed. Data in the Transmit FIFO is discarded.
1	clr_rx_err	Assertion ("1") causes the Receive FIFO to be flushed. Data in the Receive FIFO is discarded.
0	rst_mac	Assertion ("1") causes the MAC to be reset.





7.0 UTOPIA Level 2

The functionality supported by the UTOPIA Level 2 interface is tightly coupled with the code written on the Network Processor Engine (NPE). This chapter details the full hardware capabilities of the UTOPIA Level 2 interface contained within the UTOPIA Level 2 coprocessor of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described in this chapter.

Note: Not all of the IXP45X/IXP46X network processors have this functionality. For details, see *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Developer's Manual*.

7.1 Introduction

UTOPIA Level 2 is an industry-standard interface that is used to provide connection between Asynchronous Transmission Mode (ATM) and physical layer (PHY) of an ATM network. The UTOPIA Level 2 coprocessor is the entity within the IXP45X/IXP46X network processors that provides the UTOPIA Level 2 interface.

The UTOPIA Level 2 coprocessor, implemented on the IXP45X/IXP46X network processors, provides an 8-bit, UTOPIA Level 2 interface operating at speeds of up to 33 MHz. The UTOPIA Level 2 interface can be configured to operate in a single-PHY (SPHY) or a multiple-PHY (MPHY) environment.

The interface contains five transmit and five receive address lines for multi-PHY selection. The UTOPIA Level 2 coprocessor is comprised of three functional modules:

- UTOPIA Transmit Module
- UTOPIA Receive Module
- Network Processor Engine (NPE) Core Interface Module

Two 128-byte-deep FIFOs are contained in each direction of data flow — one FIFO for transmit and one FIFO for receive. Each FIFO is organized into two cell buffers, each being 64 bytes deep. This FIFO arrangement allows the receive module to be processing a cell and storing it away at the same time the Network Processor Engine core is processing a previously received cell.

In the transmit direction, the Network Processor Engine core can be placing a cell into one transmit buffer while the Transmit Module is removing the cell from the other transmit buffer.

When operating in single-PHY (SPHY) mode, the UTOPIA Level 2 interface will support octet- or cell-level handshaking as defined by the UTOPIA Level 2 specification. When configured in multiple-PHY (MPHY) mode, only cell-level handshaking is supported.

The hardware interface allows connection of up to 31 physical interface devices, as defined in the UTOPIA Level 2 specification. However, the ATM Adaptation Layers (AAL) supports only the following number of physical devices:



- IXP45X/IXP46X network processors — 31 physical devices

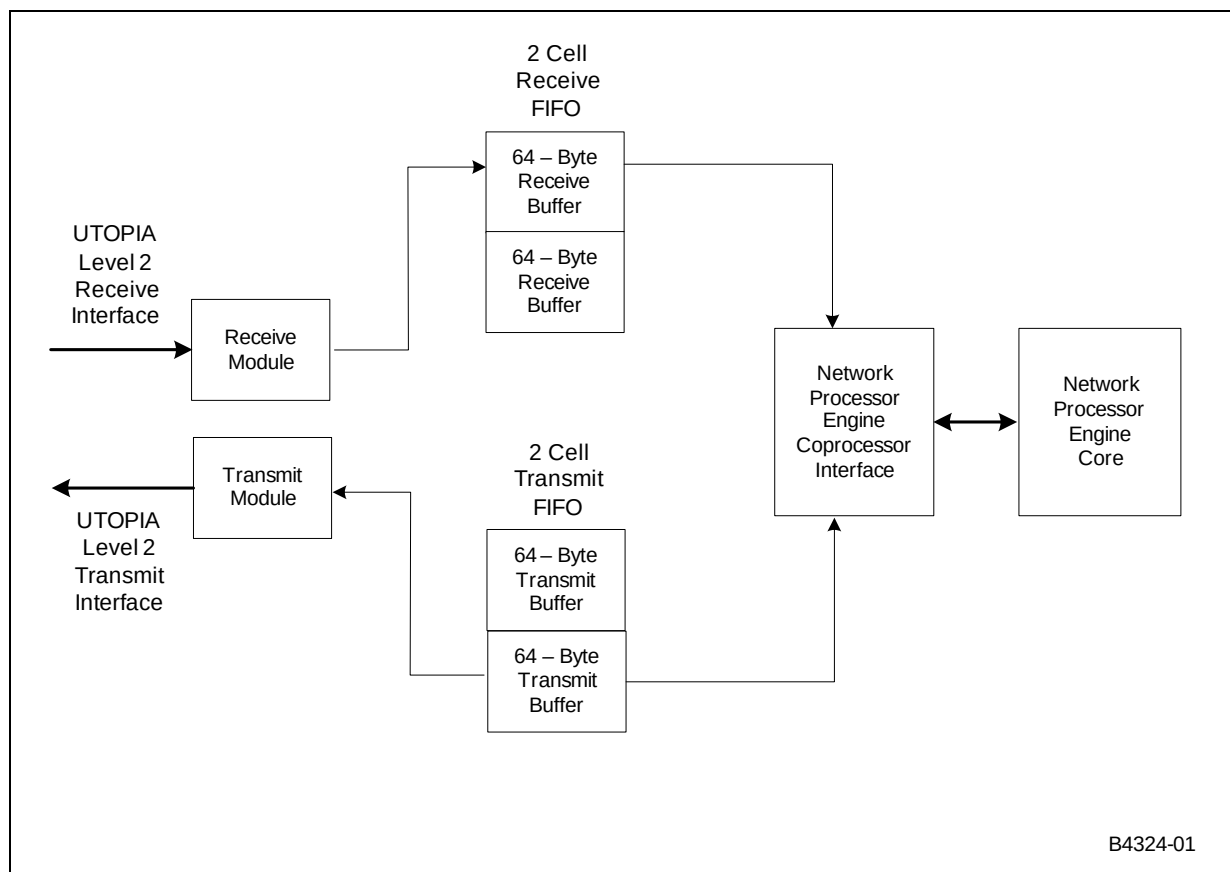
Additionally, the two-cycle polling routine defined by the UTOPIA Level 2 specification limits the number of physical devices to 26.

The ATM Adaptation Layers (AAL) implemented by the Network Processor Engine may have some further limitations on the number of physical interfaces supported. For more details on the number of physical interfaces supported, see the *Intel® IXP400 Software Programmer's Guide*.

On the Network Processor Engine side, the UTOPIA Level 2 coprocessor interfaces to the Network Processor Engine Core via the Network Processor Engine Coprocessor Bus Interface. The Network Processor Engine Coprocessor Bus Interface is used to transfer data to and from the Network Processor Engine core. The Network Processor Engine Coprocessor Bus Interface also is used to access status and configuration information for the UTOPIA Level 2 coprocessor.

Figure 32 shows the various modules within the UTOPIA Level 2 coprocessor.

Figure 32. UTOPIA Level 2 Coprocessor



7.2 UTOPIA Transmit Module

The functionality supported by the Transmit Module is tightly coupled with the code written on the Network Processor Engine. This section details the full hardware capabilities of the Transmit Module contained within the UTOPIA Level 2 coprocessor of



the IXP45X/IXP46X network processors' control plane processors. The module's user-accessible features are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described in this section.

The UTOPIA Level 2 Transmit interface transfers ATM cells to one or more UTOPIA-compliant physical devices. In multiple-PHY (MPHY) mode, the UTOPIA Level 2 transmit interface uses a round-robin polling routine to poll the various physical interfaces using the five transmit address lines (UTP_OP_ADDR) to determine which physical interfaces are ready to accept data transfers. The result of the polling is provided as status to the Network Processor Engine.

The Transmit Module is the entity, within the UTOPIA coprocessor, that implements this functionality. The Transmit Module will poll a programmable number of physical interfaces as defined by the Transmit Address Range (TXADDRRANGE) register.

If five physical interfaces are connected to the UTOPIA Level 2 interface, a value of four can be programmed into the Transmit Address Range (TXADDRRANGE) register by the Network Processor Engine Core. The polling will always begin at logic address 0 and poll sequentially to the value contained in the Transmit Address Range (TXADDRRANGE) register.

To allow the most flexibility, a logical address to physical address table is provided. The look-up table makes it possible for the five addresses — that were called out, above — not to be in sequential order. For example, the following logical to physical address map could be used for the above example of five physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_OP_ADDR lines = 00011
- Logical Address 1 => Physical Address 5 => UTP_OP_ADDR lines = 00101
- Logical Address 2 => Physical Address 7 => UTP_OP_ADDR lines = 00111
- Logical Address 3 => Physical Address 9 => UTP_OP_ADDR lines = 01001
- Logical Address 4 => Physical Address 22 => UTP_OP_ADDR lines = 10110

Once the physical address is driven to all physical interfaces, using the UTP_OP_ADDR signals, the physical interface is ready to accept a cell. The physical interface is configured to the address signals that match the values contained on the UTP_OP_ADDR signals and responds to the UTOPIA Level 2 interface on the IXP45X/IXP46X network processors by driving the UTP_OP_FCI (a.k.a TX_FULL_N/TX_CLAV) signal to inform the UTOPIA Level 2 Interface that the physical interface is ready to receive a cell.

The Transmit Port Status (TXPORTSTAT) register — contained within the Transmit Module — stores the polling result for each of the physical interfaces. The Network Processor Engine core uses the values stored in the Transmit Port Status (TXPORTSTAT) Register to select a physical interface that is ready to complete a transfer and loads the Transmit FIFO.

The Transmit FIFO informs the Transmit Module that a cell is ready to be transmitted to a specific physical interface. The Transmit Module will then remove the cell information from the Transmit FIFO and begin transmitting the data to the specified physical interface. It is important to note that the NPE code will send to the Transmit Module the logical port address of the physical interface to be selected along with the cell data. This feature allows the NPE to have full control over transmitted data based upon the polling status returned from the hardware.

While transmitting the data, an optional head-error correction (HEC) value can be calculated from the header and inserted into the data stream. The HEC generation unit takes the header data and uses the data with an internal, 8-bit HEC cyclical redundancy



check (CRC) residue register to produce the value for the next HEC CRC residue. The HEC is generated new for every cell transmitted and has no dependencies on previous cells transmitted.

The HEC residue may be inserted directly into the data stream being transmitted over the UTOPIA Level 2 interface or, optionally, the HEC residue may be exclusive-ORed with hexadecimal 0x55 — to generate a COSET value, before being inserted into the data stream.

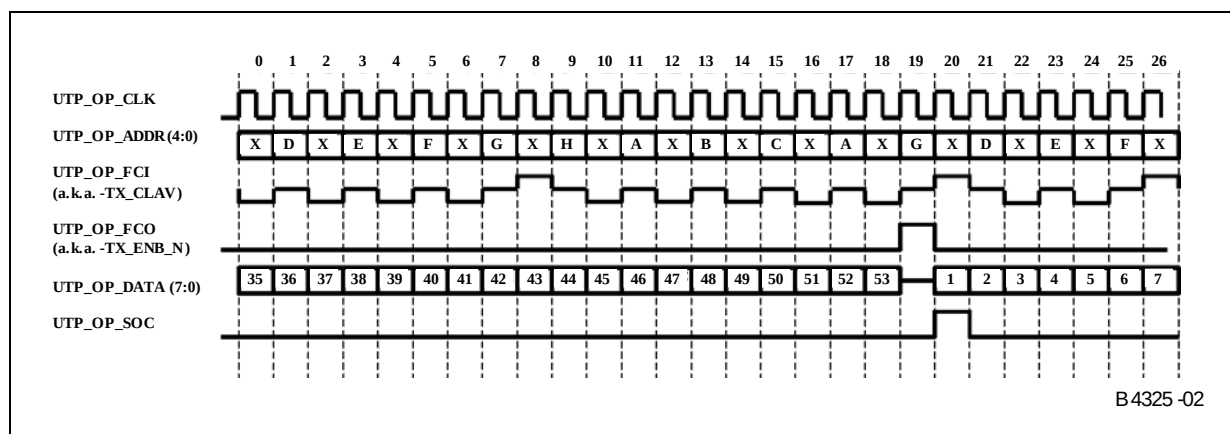
The HEC value is available one clock period after the last byte of the header information is transmitted. Therefore, a complete stream of cell data (H0, H1, H2, H3, HEC, D0, D1, ...) can be transmitted in successive clock cycles without interruption to the data stream.

When the Transmit HEC (TxHEC) configuration bit is enabled, the UTOPIA transmit interface will always insert an extra byte (valid HEC) into the cell being transmitted. In normal operation (TxHEC is enabled), the UTOPIA 2 Coprocessor Transmit Module will expect 52 bytes from the Transmit FIFO and the Transmit Module will insert a valid HEC field into the data stream.

Figure 33 shows the transmission of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for the figure:

- There are eight active physical interfaces connected, named A through H, that map to logical address 0 through 7.
- Physical Interface A is the currently selected physical interface for clock cycles 0 through 18.
- Notice on clock 8 that the result from Physical Interface G is that Physical Interface G is ready to receive a cell. The UTP_OP_FCI signal flags that a full cell can be sent to Physical Interface G by the Physical Interface asserting the UTP_OP_FCI to logic 1 one clock after Physical Interface G has been polled.
- On clock 17, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that was polled previously prior to this location.
- Notice on clock cycles 19 and 20 that Physical Interface G is selected as the next Physical Interface that the IXP45X/IXP46X network processors will transmit data to.

Figure 33. UTOPIA Level 2 MPHY Transmit Polling





In cell-level single-PHY (SPHY) mode the physical interface indicates that it can accept a cell by asserting the UTP_OP_FCI (also known as TX_FULL_N/TX_CLAV) signal. The UTOPIA Level 2 Interface subsequently transmits a cell to the PHY at the same time asserting UTP_OP_FCO (also known as TX_ENB_N).

For more timing diagrams and more details on operation in single-PHY (SPHY) mode of operation, see the *UTOPIA Level 2, Revision 1.0 Specification*.

When using the UTOPIA Level 2 Interface in single-PHY (SPHY) mode the UTP_OP_ADDR is driven with an address of all logic 1s and the UTP_IP_ADDR must be driven with all logic 1s.

In octet-level single-PHY (SPHY) mode, the physical interface indicates to the UTOPIA Level 2 Interface on the IXP45X/IXP46X network processors that the physical interface can accept data by de-asserting UTP_OP_FCI (also known as TX_FULL_N/TX_CLAV) signal. The UTOPIA Level 2 Interface subsequently transmits data to the PHY at the same time, asserting UTP_OP_FCO (also known as TX_ENB_N).

When the physical interface de-asserts UTP_OP_FCI (also known as TX_FULL_N/TX_CLAV), it indicates to the UTOPIA Level 2 Interface that the physical interface will only accept four more bytes of data.

For more timing diagrams and details on the single-PHY (SPHY) mode of operation, see the *UTOPIA Level 2, Revision 1.0 Specification*.

In addition to supporting data transmission and HEC generation, the Transmit Module maintains some statistical values. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters will maintain the following counts:

- The number of cells transmitted
- The number of idle cells transmitted

The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values. There is an overflow bit for each counter to indicate that the count has “rolled-over.” A mask-able interrupt mechanism is used to enable the UTOPIA Level 2 coprocessor to flag to the Network Processor Engine core that the “roll over” has occurred.

7.3 UTOPIA Receive Module

The functionality supported by the Receive Module is tightly coupled with the code written on the Network Processor Engine core. This section details the full hardware capabilities of the Receive Module contained within the UTOPIA Level 2 Coprocessor of the IXP45X/IXP46X network processors. The module’s user-accessible features are described in the *Intel® IXP400 Software Programmer’s Guide* and may be a subset of the features described in this section.

The UTOPIA Level 2 Receive interface receives ATM cells from one or more UTOPIA-compliant physical devices.

In multiple-PHY (MPHY) mode, the UTOPIA Level 2 receive interface uses a round-robin polling routine to poll the various physical interfaces using the five receive address lines (UTP_IP_ADDR) to determine which physical interfaces are ready to send data. The result of the polling is provided as status to the Network Processor Engine core. The Receive Module is the entity within the UTOPIA coprocessor that implements this functionality.



The Receive Module will poll a programmable number of physical interfaces, as defined by the Receive Address Range (RXADDRRANGE) register. If three physical interfaces are connected to the UTOPIA Level 2 interface, a value of two can be programmed into the Receive Address Range (RXADDRRANGE) register by the Network Processor Engine core. The polling will always begin at address 0 and poll sequentially to the value contained in the Receive Address Range (RXADDRRANGE) register. If, for example, a two was programmed into the Receive Address Range (RXADDRRANGE) register, the external physical interfaces would have to be configured to respond to the first three physical addresses produced by the UTOPIA Level 2 UTP_IP_ADDR signals.

To allow the most flexibility a logical address to physical address table is provided. The look-up table makes it possible for the three addresses that were called out above not to be in sequential order.

For example, the following logical to physical address map could be used for the above example of three physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_IP_ADDR lines = "00011"
- Logical Address 1 => Physical Address 5 => UTP_IP_ADDR lines = "00101"
- Logical Address 2 => Physical Address 7 => UTP_IP_ADDR lines = "00111"

Once the physical address is driven to all physical interfaces using the UTP_IP_ADDR signals. The physical interface that is prepared to send a cell — and configured to the address signals that match the values contained on the UTP_IP_ADDR signals — responds to the UTOPIA Level 2 Interface on the IXP45X/IXP46X network processors by driving the UTP_IP_FCI (also known as RX_EMPTY_N/RX_CLAV) signal, to inform the UTOPIA Level 2 Interface that the physical interface is ready to send a cell.

The Receive Port Status (RXPORTSTAT) register, contained within the Receive Module, stores the polling result for each of the physical interfaces. The UTOPIA Level 2 hardware uses the values — stored in the Receive Port Status (RXPORTSTAT) Register — to determine the physical interface the received cell originated from. The Receive Module will store the received cell along with the physical interface address that the cell was received from into the Receive FIFO with some basic filtering capability if desired. The Network Processor Engine will then read the address from which the cell originated and then the cell.

The Receive Module performs an optional, cell-level filtering that may cause a cell to be discarded prior to being placed into the Receive FIFO. Some of these features can be enabled or disabled include:

- Received cells that are too short are discarded
- Excess bytes of received cells that are too long are discard
- Detection of HEC errors in the cell header causes the cell to be discarded (Can be enabled/disabled)
- Detection of idle cells will be discarded (Can be enabled/disabled. The definition of an idle cell is programmable by setting the appropriate values in the Receive Define Idle [RxDefineIdle] registers)

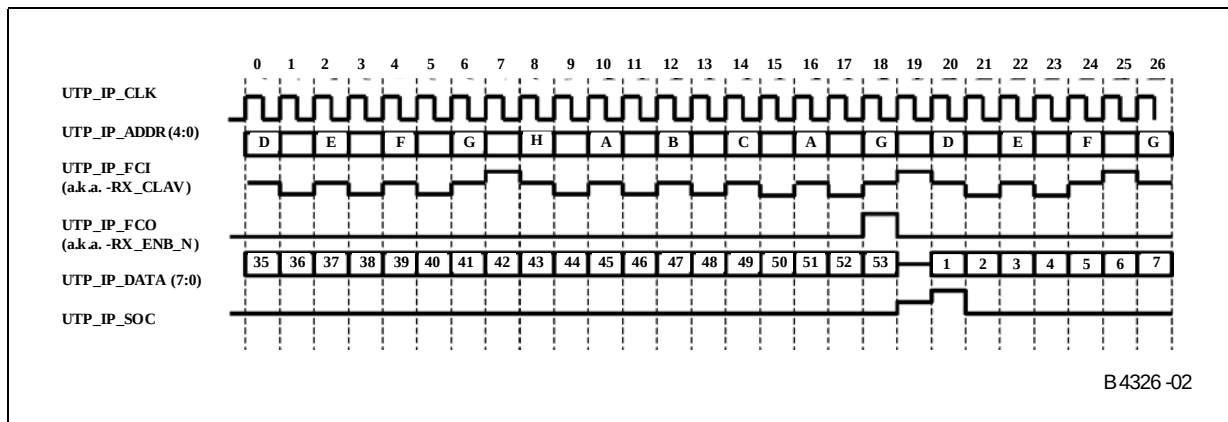
As the Receive Module is placing data into the Receive FIFO, the header information is being passed to the Receive Pre-Hash Unit.

The UTOPIA Coprocessor Receive Pre-Hash Module provides a mechanism for allowing incoming UTOPIA cells to have the ATM header looked up in a hash table to achieve faster address recognition. The hash unit takes the incoming ATM header, combines it with the arriving port information, and produces a header that can be read by the Network Processor Engine core. The Receive Pre-Hash Function can be enabled or disabled and can be used in single-PHY or multiple-PHY modes of operation. When used in single-PHY mode of operation, the port address will always be zero.

Figure 34 shows the reception of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for the figure:

- There are eight active physical interfaces connected named A through H which map to logical address 0 through 7.
- Physical Interface A is the currently selected physical interface for clock cycles 0 through 18.
- Notice on clock 7 that the result from Physical Interface G is that Physical Interface G has a full cell ready for the IXP45X/IXP46X network processors. The UTP_IP_FCI signal flags that a full cell is ready to be sent by Physical Interface G to the IXP45X/IXP46X network processors, asserting the UTP_IP_FCI to logic 1 one clock after Physical Interface G has been polled.
- On clock 16, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that was polled previously prior to this location.
- Notice on clock cycles 18 and 19 that Physical Interface G is selected as the next Physical Interface that the IXP45X/IXP46X network processors will receive data from that Physical Interface.

Figure 34. UTOPIA Level 2 MPHY Receive Polling



In cell-level single-PHY (SPHY) mode, the physical interface indicates that a cell is ready to be sent by asserting the UTP_IP_FCI (a.k.a. RX_EMPTY_N/RX_CLAV) signal. The UTOPIA Level-2 Interface on the IXP45X/IXP46X network processors subsequently initiates the transfer of a cell from the physical interface by asserting UTP_IP_FCO (a.k.a. RX_ENB_N).

In octet-level single-PHY (SPHY) mode, the UTOPIA Level 2 Interface on the IXP45X/IXP46X network processors indicates to the physical interface that the UTOPIA Receive interface is ready to receive bytes by asserting UTP_IP_FCO (a.k.a. RX_ENB_N) signal. The physical interface indicates a valid byte is on the UTOPIA data bus by de-asserting UTP_IP_FCI (also known as RX_EMPTY_N/RX_CLAV) signal.

The Receive Module maintains various statistical counters. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters will maintain the following counts:

- The number of cells received
- The number of cells with an incorrect cell size
- The number of cells containing HEC errors.
- The number of idle cells received



The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values.

There is an overflow bit per counter to indicate that the count has “rolled over.” A maskable interrupt mechanism is used to allow the UTOPIA Level 2 Coprocessor to flag to the Network Processor Engine Core that a “roll over” has occurred.

7.4 UTOPIA Level 2 Coprocessor / NPE Coprocessor: Bus Interface

The Network Processor Engine Coprocessor Interface Module provides the necessary interface logic required for configuration, monitoring, control, and test of the UTOPIA 2 coprocessor. All of the UTOPIA 2 coprocessor's internal configuration and control registers, instruction registers, and FIFOs are directly accessible by the Network Processor Engine core.

7.5 MPHY Polling Routines

The UTOPIA 2 coprocessor implements a round-robin polling algorithm. The Receive and Transmit modules use a logical-to-physical address-translation table to determine the actual physical interface that is to be polled. This feature allows the designer complete control over the physical address polling sequence.

The multiple-PHY (MPHY) address translation is used by the UTOPIA Level 2 Interface, on the IXP45X/IXP46X network processors, to poll physical addresses that are not contiguous or do not start at 0.

There are two translation tables implemented. One translation table is used for receive interface polling and the other translation table is used for transmit interface polling. Each translation table is implemented as 31 5-bit registers. Each register is addressed from 0 to 30, corresponding to one of 31 logical addresses.

The five bits of each register are used to designate a physical interface number. Therefore if a binary value of 00101 is written to address location 0 (logical port 0) of the transmit translation table, the polling sequence would actually assert a five on the transmit (UTP_OP_ADDR) address lines of the UTOPIA Level 2 interface during logical port 0's turn in the polling algorithm.

For example, make the following assumptions:

- A design requires eight physical interfaces to be connected, which are configured to respond to addresses 0 through 7.
- The polling order of the physical interfaces is required to be 1, 3, 5, 7, 0, 2, 4, and 6 for both transmit and receive.

To accomplish this polling sequence:

1. The Network Processor Engine core will set the TXADDRRANGE and the RXADDRRANGE to a hexadecimal value of 0x7.
This will identify that there are eight physical interfaces attached and involved in the polling sequence.
2. Define the values in both the transmit and receive translation tables as follows:
 - Address 0 (logical address 0) = A binary value of 00001
 - Address 1 (logical address 1) = A binary value of 00011
 - Address 2 (logical address 2) = A binary value of 00101
 - Address 3 (logical address 3) = A binary value of 00111



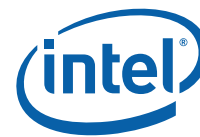
- Address 4 (logical address 4) = A binary value of 00000
- Address 5 (logical address 5) = A binary value of 00010
- Address 6 (logical address 6) = A binary value of 00100
- Address 7 (logical address 7) = A binary value of 00110

The polling sequence at the physical interface will rotate from logical address 0 through 7. This event will cause the physical address polling values on the UTOPIA Level 2 physical interface to be 1, 3, 5, 7, 0, 2, 4, 6.

7.6 UTOPIA Level 2 Clocks

The UTOPIA Level 2 interface on the IXP45X/IXP46X network processors characterizes the interface for clock speeds of 25 MHz and 33 MHz.

The UTOPIA Level 2 interface requires both transmit and receive clock inputs to be supplied from an external source. The transmit module and receive module of the UTOPIA Level 2 interface can have independent clocks running at separate clock speeds.





8.0 USB 1.1 Device Controller

This chapter describes the Universal Serial Bus (USB) protocol and its implementation-specific options for device controllers. These options include:

- Endpoint number, type, and function
- Interrupts to the Intel XScale® Processor
- A transmit/receive FIFO interface

A working knowledge of the USB standard is vital to effective use of this chapter. The Universal Serial Bus Device Controller (UDC) of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors is USB-compliant and supports all standard device requests issued by the host. UDC operation summaries and quick reference tables are provided.

Refer to the *Universal Serial Bus Specification*, revision 1.1, for a full description of the USB protocol. The *Universal Serial Bus Specification* is available at the following Web site:

<http://www.usb.org>.

Both this, USB-device interface and the USB-host interface can be used at the same time. (For more information on the USB-host interface, see “USB 2.0 Host Controller” on page 352.)

8.1 USB Overview

The UDC supports 16 endpoints and can operate half-duplex at a rate of 12 Mbps (as a slave only, not as a host or hub controller). The UDC supports four device configurations. Configurations 1, 2, and 3 each support two interfaces. This allows the host to accommodate dynamic changes in the physical bus topology. A configuration is a specific combination of USB resources available on the device. An interface is a related set of endpoints that present a device feature or function to the host.

The UDC transmits serial information that contains layers of communication protocols. Fields are the most basic protocol. UDC fields include:

- Sync
- Packet identifier (PID)
- Address
- Endpoint
- Frame number
- Data
- Cyclic Redundancy Check (CRC)

Fields are combined to produce packets.

A packet's function determines the combination and number of fields that make up the packet. Packet types include:

- Token
- Start of frame
- Data
- Handshake



Packets are assembled into groups to produce transactions. Transactions fall into four groups:

- Bulk
- Control
- Interrupt
- Isochronous

Endpoint 0 is used only to communicate the control transactions that configure the UDC. Endpoint 0s responsibilities include:

- Connection
- Address assignment
- Endpoint configuration
- Bus enumeration
- Disconnection

The UDC uses a dual-port memory to support FIFO operations. Each Bulk and Isochronous Endpoint FIFO structure is double-buffered to enable the endpoint to process one packet as it assembles another. The Intel XScale processor can fill and empty the FIFOs. An interrupt is generated when a packet has been received. Interrupts are also generated when the FIFO encounters a short packet or zero-length packet. Endpoint 0 has a 16-entry-long, 8-bit-wide FIFO that can only be read or written by the Intel XScale processor.

For endpoints 1-15, the UDC uses its dual-ported memory to hold data for a Bulk OUT transaction while the transaction is checked for errors. If the Bulk OUT transaction data is invalid, the UDC sends a NAK handshake to request the host to resend the data. The software is not notified that the OUT data is invalid until the Bulk OUT data is received and verified. If the host sends a NAK handshake in response to a Bulk IN data transmission, the UDC resends the data. Because the FIFO maintains a copy of the data, the software does not have to reload the data.

The external pins dedicated to the UDC interface are UDC+ and UDC-. The USB protocol uses differential signalling between the two pins for half-duplex data transmission. A 1.5-K Ω pull-up resistor must be connected to the USB cable's D+ signal to pull the UDC+ pin high when it is not driven. Pulling the UDC+ pin high when it is not driven allows the UDC to be a high-speed, 12-Mbps device and provides the correct polarity for data transmission.

The serial bus uses differential signalling to transmit multiple states simultaneously. These states are combined to produce transmit data and various bus conditions, including:

- Idle
- Resume
- Start of Packet
- End of Packet
- Disconnect
- Connect
- Reset

8.2 Device Configuration

The configuration of the Universal Serial Bus 1.1 Device Controller is shown in [Table 107](#).



Table 107. Endpoint Configuration: Universal Serial Bus Device Controller

Endpoint Number	Type	Function	FIFO Size (bytes) X Number of FIFOs
0	Control	IN/OUT	16
1	Bulk	IN	64x2
2	Bulk	OUT	64x2
3	Isochronous	IN	256x2
4	Isochronous	OUT	256x2
5	Interrupt	IN	8
6	Bulk	IN	64x2
7	Bulk	OUT	64x2
8	Isochronous	IN	256x2
9	Isochronous	OUT	256x2
10	Interrupt	IN	8
11	Bulk	IN	64x2
12	Bulk	OUT	64x2
13	Isochronous	IN	256x2
14	Isochronous	OUT	256x2
15	Interrupt	IN	8

Data flow is relative to the USB host. IN packets represent data flow from the UDC to the USB host. OUT packets represent data flow from the USB host to the UDC.

The FIFOs for the bulk and isochronous endpoints are double-buffered so that one packet can be processed while the next is being assembled. While the UDC is busy transmitting an IN packet from a particular endpoint, the Intel XScale processor can be loading the same endpoint for the next frame transmission. Likewise, while the Intel XScale processor is unloading an OUT endpoint, the UDC can continue to process the next incoming packet to that endpoint.

8.3 USB Operation

After an Intel XScale processor reset — or when the USB host issues a USB reset — the UDC configures all endpoints and is forced to use the USB default address, 0. After the UDC configures the endpoints, the USB host assigns the UDC a unique address. At this point, the UDC is under the USB host's control and responds to commands that use control transactions to transmit to endpoint 0.

The following sections provide details of the USB protocol in a bottom-up fashion, starting with signalling levels.

8.3.1 Signalling Levels

USB uses differential signalling to encode data and to indicate various bus conditions. The USB specification refers to the J and K data states to differentiate between high- and low-speed transmission. Because the UDC supports only 12-Mbps transmission, references are made only to actual data state 0 and actual data state 1.



By decoding the polarity of the UDC+ and UDC- pins and using differential data, four distinct states are represented. Two of the four states are used to represent data. A 1 indicates that UDC+ is high and UDC- is low. A 0 indicates that UDC+ is low and UDC- is high. The two remaining states and pairings of the four encodings are further decoded to represent the current state of the USB.

Table 108 shows how differential signalling represents eight different bus states.

Table 108. USB States

Bus State	UDC+ / UDC- Pin Levels
Idle	UDC+ high, UDC- low (same as a 1).
Suspend	Idle state for more than 3 ms.
Resume	UDC+ low, UDC- high (same as a 0).
Start of Packet	Transition from idle to resume.
End of Packet	UDC+ AND UDC- low for 2-bit times followed by an idle for 1-bit time.
Disconnect	UDC+ AND UDC- below single-ended low threshold for more than 2.5 μ s. (Disconnect is the static bus condition that results when no device is plugged into a hub port.)
Connect	UDC+ OR UDC- high for more than 2.5 μ s.
Reset	UDC+ AND UDC- low for more than 2.5 μ s. (Reset is driven by the USB host controller and sensed by a device controller.)

USB Hosts and USB hubs have pull-down resistors on both the D+ and D- lines. When a device is not attached to the cable, the pull-down resistors cause D+ and D- to be pulled down below the single-ended low threshold of the USB host or USB hub. This creates a state called single-ended zero (SE0). A disconnect is detected by the USB host when an SE0 persists for more than 2.5 μ s (30-bit times). When the UDC is connected to the USB cable, the pull-up resistor on the UDC+ pin causes D+ to be pulled above the single-ended high threshold level. After 2.5 μ s elapse, the USB host detects a connect.

After the USB Host detects a Connect, the bus is in the idle state because UDC+ is high and UDC- is low. The bus transitions from the Idle state to the Resume state (a 1-to-0 transition) to signal the Start of Packet (SOP). Each USB packet begins with a Sync field that starts with the 1-to-0 transition.

After the packet data is transferred, the bus signals the End of Packet (EOP) state by pulling both UDC+ and UDC- low for 2 bit times, followed by an Idle state for 1 bit time. If the idle persists for more than 3 ms, the UDC enters Suspend state. The USB Host can awaken the UDC from the Suspend state by signalling a reset or by switching the bus to the resume state via normal bus activity. Under normal operating conditions, the USB host periodically signals an Start of Frame (SOF) to ensure that devices do not enter the suspend state.

8.3.2 Bit Encoding

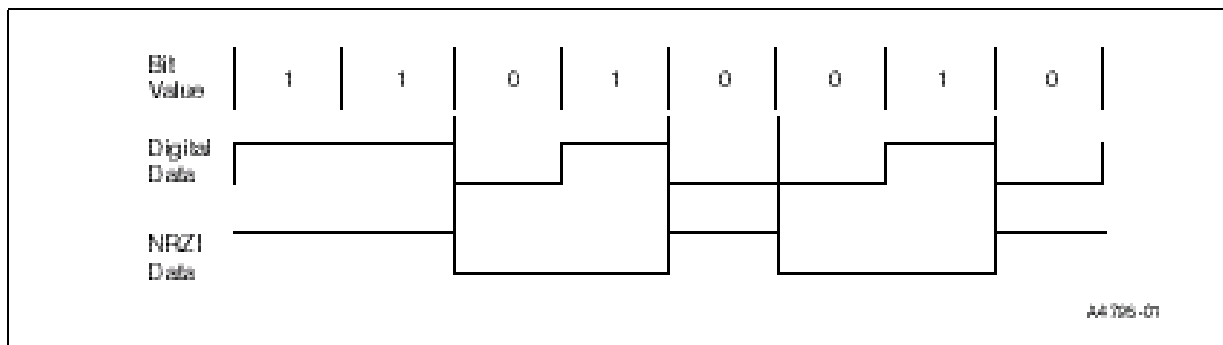
USB uses non-return to zero inverted (NRZI) to encode individual bits. Both the clock and the data are encoded and transmitted within the same signal. Instead of representing data by controlling the state of the signal, transitions are used. A 0 is represented by a transition, and a 1 is represented by no transition (this produces the data).



Each time a 0 occurs, the receiver logic synchronizes the baud clock to the incoming data (thus producing the clock). To ensure the receiver is periodically synchronized, any time six consecutive 1s are detected in the serial bit stream, a 0 is automatically inserted by the transmitter. This procedure is known as “bit stuffing.” The receiver logic, in turn, automatically detects stuffed bits and removes them from the incoming data.

Bit stuffing causes a transition on the incoming signal at least once every 7 bit-times to guarantee baud-clock lock. Bit stuffing is enabled for an entire packet beginning when the start of packet is detected and maintained until the end of packet is detected (enabled during the sync field all the way through the CRC field). Figure 35 shows the NRZI encoding of the data byte 0b1101 0010.

Figure 35. NRZI Bit Encoding Example



8.3.3 Field Formats

Individual bits are assembled into groups called fields. Fields are used to construct packets and packets are used to construct frames or transactions. There are seven USB field types:

- Sync
- PID
- Address
- Endpoint
- Frame Number
- Data
- CRC

A Sync is preceded by the Idle state and is the first field of every packet. The first bit of a Sync field signals the SOP to the UDC or host. A Sync is 8 bits wide and consists of seven 0s, followed by a 1 (0x80). Bits are transmitted to the bus least-significant-bit-first in every field, except the CRC field.

The PID is 1 byte wide and always follows the sync field. The first four bits contain an encoded value that represents packet type (Token, Data, Handshake, and Special), packet format, and type of error detection. The last four bits contain a check field that ensures the PID is transmitted without errors.

The check field is generated by performing a 1-s complement of the PID. The UDC XORs the PID and CRC fields and takes the action prescribed in the USB standard if the result does not contain all ones, which indicates an error has occurred in transmission.

The Address and Endpoint fields are used to access the UDC's 16 endpoints. The Address field contains seven bits and permits 127 unique devices to be placed on the USB. After the USB host signals a reset, the UDC and all other devices are assigned the default address, 0.



The host is then responsible for assigning a unique address to each device on the bus. Addresses are assigned in the enumeration process, one device at a time. After the host assigns the an address to the UDC, the UDC only responds to transactions directed to that address. The Address field follows the PID in every packet transmitted.

When the UDC detects a packet that is addressed to it, it uses the Endpoint field to determine which of the UDC's endpoints is being addressed. The Endpoint field contains four bits. Encodings for endpoints 0 (0000b) through 15 (1111b) are allowed. The Endpoint field follows the Address field. [Table 109](#) shows the valid values for the endpoint field.

Table 109. Endpoint Field Addressing

Endpoint Field Value	UDC Endpoint Selected
0000	Endpoint 0
0001	Endpoint 1
0010	Endpoint 2
0011	Endpoint 3
0100	Endpoint 4
0101	Endpoint 5
0110	Endpoint 6
0111	Endpoint 7
1000	Endpoint 8
1001	Endpoint 9
1010	Endpoint 10
1011	Endpoint 11
1100	Endpoint 12
1101	Endpoint 13
1110	Endpoint 14
1111	Endpoint 15

The Frame Number is an 11-bit field incremented by the host each time a frame is transmitted. When it reaches its maximum value (2,047 [0x7FF]) its value rolls over. Frame Number is transmitted in the SOF packet, which the host outputs in 1-ms intervals.

Device controllers use the Frame Number field to control isochronous transfers. Data fields are used to transmit the packet data between the host and the UDC. A data field consists of 0 to 1,023 bytes. Each byte is transmitted least-significant-bit-first. The UDC generates an interrupt to indicate that a Start of Frame event has occurred.

CRC fields are used to detect errors introduced during token and data-packet transmission and are applied to all the fields in the packet except the PID field. The PID contains its own 4-bit 1s complement check field for error detection. Token packets use a 5-bit CRC ($x^5 + x^2 + 1$) called CRC5 and Data packets use a 16-bit CRC ($x^{16} + x^{15} + x^2 + 1$) called CRC16. For both CRCs, the checker resets to all 1s at the start of each packet.

8.3.4 Packet Formats

USB supports four packet types:



- Token
- Handshake
- Data
- Special

A PRE (Preamble) PID precedes a low-speed (1.5 Mbps) USB transmission. The UDC supports high-speed (12 Mbps) USB transfers only. PRE packets that signify low-speed devices and the low-speed data transfer that follows such PRE packets are ignored.

8.3.4.1 Token Packet Type

A token packet is placed at the beginning of a frame and is used to identify OUT, IN, SOF, and SETUP transactions. OUT and IN frames are used to transfer data, SOF packets are used to time isochronous transactions, and SETUP packets are used for control transfers to configure endpoints.

A Token packet consists of a sync, a PID, an address, an endpoint, and a CRC5 field (see [Table 110](#)). For OUT and SETUP transactions, the address and endpoint fields are used to select the UDC endpoint that receives the data. For an IN transaction, the address and endpoint fields are used to select the UDC endpoint that transmits data.

Table 110. IN, OUT, and SETUP Token Packet Format

8 Bits	8 Bits	7 Bits	4 Bits	5 Bits
Sync	PID	Address	Endpoint	CRC5

8.3.4.2 Start-of-Frame Packet Type

A Start-of-Frame (SOF) packet is a special type of token packet that is issued by the USB host at a nominal interval of once every 1 ms +/- 0.0005 ms.

SOF packets consist of a sync, a PID, a frame number (which is incremented after each frame is transmitted), and a CRC5 field, as shown in [Table 111](#). The presence of SOF packets every 1 ms prevents the UDC from going into suspend mode.

Table 111. SOF Token Packet Format

8 Bits	8 Bits	11 Bits	5 Bits
Sync	PID	Frame Number	CRC5

8.3.4.3 Data Packet Type

Data packets follow token packets and are used to transmit data between the USB host and UDC. There are two types of data packets as specified by the PID: DATA0 and DATA1. These two types of Data Packets are used to provide a mechanism to guarantee data sequence synchronization between the transmitter and receiver across multiple transactions.

During the handshake phase, the transmitter and receiver determine which data token type to transmit first. For each subsequent packet transmitted, the data packet type is toggled (DATA0, DATA1, DATA0, and so on).

A data packet consists of a sync; a PID; from 0 to 1,023 bytes of data; and a CRC16 field, as shown in [Table 112](#). Note that the UDC supports a maximum of 8 bytes of data for an Interrupt IN data payload, a maximum of 64 bytes of data for a bulk data payload and a maximum of 256 bytes of data for an isochronous data payload.



Table 112. Data Packet Format

8 Bits	8 Bits	0 - 1,023 Bytes	16 Bits
Sync	PID	Data	CRC16

8.3.4.4 Handshake Packet Type

Handshake packets consist of a Sync and a PID. Handshake packets do not contain a CRC because the PID contains its own check field. Handshake packets are used to report data transaction status, including confirmation that data was successfully received, flow control, and stall conditions. Only transactions that support flow control can return handshakes.

The three types of handshake packets are: ACK, NAK, and STALL.

ACK indicates that a data packet was received without bit stuffing, CRC, or PID check errors. NAK indicates that the UDC was unable to accept data from the host or has no data to transmit. STALL indicates that the UDC was unable to transmit or receive data, and requires host intervention to clear the stall condition.

The receiving unit signals Bit stuffing, CRC, and PID errors by omitting a handshake packet. [Table 113](#) shows the format of a handshake packet.

Table 113. Handshake Packet Format

8 Bits	8 Bits
Sync	PID

8.3.5 Transaction Formats

Packets are assembled into groups to form transactions. The USB protocol uses four different transaction formats. Each transaction format is specific to a particular type of endpoint: Bulk, Control, Interrupt, or Isochronous.

Endpoint 0, by default, is a control endpoint and receives only control transactions. All USB transactions are initiated by the host controller and transmitted in one direction at a time (known as half-duplex) between the host and UDC.

8.3.5.1 Bulk Transaction Type

Bulk transactions guarantee error-free data transmission between the host and UDC by using packet-error detection and retry. The host schedules bulk packets when the bus has available time.

Bulk transactions are made up of three packet types: Token, Data, and Handshake. The eight types of bulk transactions are based on data direction, error, and stall conditions.

The types of bulk transactions are shown in [Table 114](#). Packets sent from the UDC to the host are highlighted in boldface type and packets sent from the host to the UDC are not.



Table 114. Bulk Transaction Formats

Action	Token Packet	Data Packet [†]	Handshake Packet [†]
Host successfully received data from UDC	IN	DATA0/DATA1	ACK
UDC temporarily unable to transmit data	IN	None	NAK
UDC endpoint needs host intervention	IN	None	STALL
Host detected PID, CRC, or bit stuff error	IN	DATA0/DATA1	None
UDC successfully received data from host	OUT	DATA0/DATA1	ACK
UDC temporarily unable to receive data	OUT	DATA0/DATA1	NAK
UDC endpoint needs host intervention	OUT	DATA0/DATA1	STALL
UDC detected PID, CRC, or bit stuff error	OUT	DATA0/DATA1	None

[†] Packets from UDC to host are **boldfaced**.

8.3.5.2 Isochronous Transaction Type

Isochronous transactions ensure constant-rate, error-tolerant transmission of data between the host and UDC. The host schedules isochronous packets during every frame.

The USB protocol allows isochronous transfers to take up to 90% of the USB bandwidth. Unlike bulk transactions, if corrupted data is received, the UDC will continue to process the corrupted data that corresponds to the current start of frame indicator.

Isochronous transactions do not support a handshake phase or retry capability. Two packet types are used to construct isochronous transactions: token and data. The types of isochronous transactions based on data direction are shown in [Table 115](#).

Table 115. Isochronous Transaction Formats

Action	Token Packet [†]	Data Packet [†]
Host received data from UDC	IN	DATA0
UDC received data from host	OUT	DATA0

[†] Packets from UDC to host are **boldfaced**.

8.3.5.3 Control Transaction Type

The host uses control transactions to configure endpoints and query their status. Like bulk transactions, control transactions begin with a setup packet, followed by an optional data packet, then a handshake packet. Control transactions, by default, use DATA0 type transfers.

[Table 116](#) shows the four types of control transactions.

Table 116. Control Transaction Formats

Action	Token Packet	Data Packet [†]	Handshake Packet [†]
UDC successfully received control from host	SETUP	DATA0	ACK
UDC temporarily unable to receive data	SETUP	DATA0	NAK
UDC endpoint needs host intervention	SETUP	DATA0	STALL
UDC detected PID, CRC, or bit stuff error	SETUP	DATA0	None

[†] Packets from UDC to host are **boldfaced**.



To assemble control transfers, the host sends a control transaction to tell the UDC what type of control transfer is taking place (control read or control write), followed by one or more data transactions. The setup is the first stage of the control transfer. The device must respond with an ACK or no handshake (if the data is corrupted).

The control transaction, by default, uses a DATA0 transfer and each subsequent data transaction toggles between DATA1 and DATA0 transfers. A control write to an endpoint uses OUT transactions.

Control reads use IN transactions. The transfer direction is the opposite of the last data transaction. The transfer direction is used to report status and functions as a handshake.

For a control write, the last transaction is an IN from the UDC to the host. For a control read, the last transaction is an OUT from the host to the UDC. The last data transaction always uses a DATA1 transfer, even if the previous transaction used DATA1.

8.3.5.4 Interrupt Transaction Type

The host uses interrupt transactions to query the status of the device. Like bulk transactions, interrupt transactions begin with a setup packet, followed by an optional data packet, then a handshake packet. Interrupt transactions, by default, use DATA0 type transfers.

Table 117 shows the four types of interrupt transactions.

Table 117. Interrupt Transaction Formats

Action	Token Packet	Data Packet [†]	Handshake Packet [†]
Host successfully received data from UDC	IN	DATA0	ACK
UDC temporarily unable to transmit data	IN	None	NAK
UDC endpoint needs host intervention	IN	None	STALL
Host detected PID, CRC, or bit stuff error	IN	DATA0	None

[†] Packets from UDC to host are **boldfaced**.

8.3.6 UDC Device Requests

The UDC uses its control, status, and data registers to control and monitor the transmit and receive FIFOs for endpoints 1 - 15. The host controls all other UDC configuration and status reporting using device requests that are sent as control transactions to endpoint 0 via the USB.

Each setup packet to Endpoint 0 is 8 bytes long and specifies:

- Data transfer direction: host to device, device to host
- Data transfer type:
 - Standard
 - Class
 - Vendor
- Data recipient:
 - Device
 - Interface
 - Endpoint
 - Other



- Number of bytes to transfer
- Index or offset
- Value: used to pass a variable-sized data parameter
- Device request

Table 118 shows a summary of all device requests. For a full description of host device requests, see the *Universal Serial Bus Specification Revision 1.1*.

Table 118. Host Device Request Summary

Request	Name
SET_FEATURE	Enables a specific feature such as device remote wake-up or endpoint stalls.
CLEAR_FEATURE	Clears or disables a specific feature.
SET_CONFIGURATION	Configures the UDC for operation. Used after a reset of the Intel XScale processor or after a reset has been signalled via the USB.
GET_CONFIGURATION	Returns the current UDC configuration to the host.
SET_DESCRIPTOR	Sets existing descriptors or add new descriptors. Existing descriptors include: • Device•Configuration • String•Interface • Endpoint
GET_DESCRIPTOR	Returns the specified descriptor, if it exists.
SET_INTERFACE	Selects an alternate setting for the UDC's interface.
GET_INTERFACE	Returns the selected alternate setting for the specified interface.
GET_STATUS	Returns the UDC's status including: remote wake-up, self-powered, data direction, endpoint number, and stall status.
SET_ADDRESS	Sets the UDC's 7-bit address value for all future device accesses.
SYNCH_FRAME	Sets then reports an endpoint's synchronization frame.

The UDC decodes most standard device commands with no intervention required by the user. The following commands are not passed to the user are:

- Set Address
- Set Feature
- Clear Feature
- Get Configuration
- Get Status
- Get Interface
- Sync Frame

The Set Configuration and Set Interface commands are passed to the user to indicate that the host set the specified configuration or interface and the software must take any necessary actions. The Get Descriptor and Set Descriptor commands are passed to the user to be decoded.

Because the Set Feature and Clear Feature commands are not passed on, the user is not able to decode the device remote wake-up feature commands. To solve this problem, the status bit UDCCS0:DRWF indicates whether or not the device remote wake-up feature is enabled. UDCCS0: DRWF is a read-only bit. When the bit is set to 1, the device remote wake-up feature is enabled. When the bit is set to 0, the feature is not enabled.

8.3.7 UDC Configuration

In response to the GET_DESCRIPTOR command, the user device sends back a description of the UDC configuration. The UDC can physically support more data channel bandwidth than the USB specification allows.



When the device responds to the host, it must specify a legal USB configuration. For example, if the device specifies a configuration of six isochronous endpoints of 256 bytes each, the host is not able to schedule the proper bandwidth and does not take the UDC out of Configuration 0. The user device determines which endpoints to report to the host. If an endpoint is not reported, it is not used.

Another option, attractive — for use with isochronous endpoints — is to describe a configuration of a packet with a maximum size less than 256 bytes to the host. For example, if software responds to the GET_DESCRIPTOR command that Endpoint 3 only supports 64 bytes maximum packet isochronous IN data, the user device must set the UDCCS3[TSP] bit after it loads 64 bytes for transmission. Similarly, if Endpoint 4 is described as supporting 128 bytes maximum packet Isochronous OUT data, the UDC recognizes the end of the packet, sets UDCCS4[RPC], and an interrupt is generated.

The direction of the endpoints is fixed. Physically, the UDC only supports interrupt endpoints with a maximum packet size of 8 bytes or less, bulk endpoints with a maximum packet size of 64 bytes or less, and isochronous endpoints with a maximum packet size of 256 bytes or less.

To make the IXP45X/IXP46X network processors more adaptable, the UDC supports a total of four configurations. Each of these configurations are identical in the UDC, software can make three distinct configurations, each with two interfaces. Configuration 0 is a default configuration of Endpoint 0 and cannot be used in any other configuration.

After the host completes a SET_CONFIGURATION or SET_INTERFACE command, the software must decode the command to empty the OUT endpoint FIFOs and allow the Intel XScale processor to set up the proper power/peripheral configurations.

8.4 UDC Hardware Connections

This section explains how to connect the USB interface for a variety of devices.

8.4.1 Self-Powered Device

The IXP45X/IXP46X network processors do not implement any low-power modes of operation. Therefore, the UDC can be connected to act as a full-speed device in a permanent configuration by pulling the USB_DPOS signal to a logic high.

The USB_DPOS signals should be pulled high by the 5-V signal received over the USB connector. The ground received from the USB connector should be tied to the ground of the board.

8.4.2 Bus-Powered Devices

The IXP45X/IXP46X network processors do not support bus-powered devices because they cannot limit the amount of current consumed to 500 μ A when the USB host issues a suspend. (Section 7.2.3 of the *USB Specification*, version 1.1 requires that less than 500 μ A is consumed when the USB host issues a suspend.)

8.5 Register Descriptions

All configuration, request/service, and status reporting is controlled by the USB host controller and is communicated to the UDC via the USB. The UDC has registers that control the interface between the UDC and the software.



A control register enables the UDC and masks the interrupt sources in the UDC. A status register indicates the state of the interrupt sources. Each of the 16 endpoints (control, OUT, and IN) have a control or status register. Endpoint 0 (control) has an address for the 16-by-8 data FIFO that can be used to transmit and receive data. Endpoint 0 also has a write count register that is used to determine the number of bytes the USB host controller has sent to Endpoint 0.

Table 119. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 120. USB-Device Register Descriptions (Sheet 1 of 2)

Address	Name	Description
0 x C800 B000	UDCCR	UDC Control Register
0 x C800 B004	(Reserved)	Reserved for Future Use
0 x C800 B008	(Reserved)	Reserved for Future Use
0 x C800 B00C	(Reserved)	Reserved for Future Use
0 x C800 B010	UDCCS0	UDC Endpoint 0 Control/Status Register
0 x C800 B014	UDCCS1	UDC Endpoint 1 (IN) Control/Status Register
0 x C800 B018	UDCCS2	UDC Endpoint 2 (OUT) Control/Status Register
0 x C800 B01C	UDCCS3	UDC Endpoint 3 (IN) Control/Status Register
0 x C800 B020	UDCCS4	UDC Endpoint 4 (OUT) Control/Status Register
0 x C800 B024	UDCCS5	UDC Endpoint 5 (Interrupt) Control/Status Register
0 x C800 B028	UDCCS6	UDC Endpoint 6 (IN) Control/Status Register
0 x C800 B02C	UDCCS7	UDC Endpoint 7 (OUT) Control/Status Register
0 x C800 B030	UDCCS8	UDC Endpoint 8 (IN) Control/Status Register
0 x C800 B034	UDCCS9	UDC Endpoint 9 (OUT) Control/Status Register
0 x C800 B038	UDCCS10	UDC Endpoint 10 (Interrupt) Control/Status Register
0 x C800 B03C	UDCCS11	UDC Endpoint 11 (IN) Control/Status Register
0 x C800 B040	UDCCS12	UDC Endpoint 12 (OUT) Control/Status Register
0 x C800 B044	UDCCS13	UDC Endpoint 13 (IN) Control/Status Register
0 x C800 B048	UDCCS14	UDC Endpoint 14 (OUT) Control/Status Register
0 x C800 B04C	UDCCS15	UDC Endpoint 15 (Interrupt) Control/Status Register
0 x C800 B050	UICR0	UDC Interrupt Control Register 0
0 x C800 B054	UICR1	UDC Interrupt Control Register 1
0 x C800 B058	UISR0	UDC Status Interrupt Register 0
0 x C800 B05C	UISR1	UDC Status Interrupt Register 1
0 x C800 B060	UFNHR	UDC Frame Number Register High
0 x C800 B064	UFNLR	UDC Frame Number Register Low



Table 120. USB-Device Register Descriptions (Sheet 2 of 2)

Address	Name	Description
0 x C800 B068	UBC2	UDC Byte Count Register 2
0 x C800 B06C	UBC4	UDC Byte Count Register 4
0 x C800 B070	UBC7	UDC Byte Count Register 7
0 x C800 B074	UBC9	UDC Byte Count Register 9
0 x C800 B078	UBC12	UDC Byte Count Register 12
0 x C800 B07C	UBC14	UDC Byte Count Register 14
0 x C800 B080	UDDR0	UDC Endpoint 0 Data Register
0 x C800 B100	UDDR1	UDC Endpoint 1 Data Register
0 x C800 B180	UDDR2	UDC Endpoint 2 Data Register
0 x C800 B200	UDDR3	UDC Endpoint 3 Data Register
0 x C800 B400	UDDR4	UDC Endpoint 4 Data Register
0 x C800 B0A0	UDDR5	UDC Endpoint 5 Data Register
0 x C800 B600	UDDR6	UDC Endpoint 6 Data Register
0 x C800 B680	UDDR7	UDC Endpoint 7 Data Register
0 x C800 B700	UDDR8	UDC Endpoint 8 Data Register
0 x C800 B900	UDDR9	UDC Endpoint 9 Data Register
0 x C800 B0C0	UDDR10	UDC Endpoint 10 Data Register
0 x C800 BB00	UDDR11	UDC Endpoint 11 Data Register
0 x C800 BB80	UDDR12	UDC Endpoint 12 Data Register
0 x C800 BC00	UDDR13	UDC Endpoint 13 Data Register
0 x C800 BE00	UDDR14	UDC Endpoint 14 Data Register
0 x C800 B0E0	UDDR15	UDC Endpoint 15 Data Register

8.5.1 UDC Control Register

(UDCCR)

The UDC control register (UDCCR) contains seven control bits: one to enable the UDC, one to show activity, and five to show status and associated control functions.

8.5.1.1 UDC Enable

The UDC Enable (UDE) bit enables the UDC. When UDE is set to a 1, the UDC is enabled for USB serial transmission or reception. When UDE is set to a 0, the UDC is disabled and the UDC+ and UDC- pins are tri-stated. This means that the UDC ignores all activity on the USB bus.

If UDE is set to a 0 the entire UDC design is reset. If the reset occurs while the UDC is actively transmitting or receiving data, it stops immediately and the remaining bits in the transmit or receive serial shifter are reset.

All entries in the transmit and receive FIFO are also reset.



8.5.1.2 UDC Active

The read-only UDC Active (UDA) bit can be read to determine if the UDC is currently active or in a USB reset. This bit is only valid when the UDC is enabled. A 0 indicates that the UDC is currently receiving a USB reset from the USB Host. A 1 indicates that the UDC is currently involved in a transaction.

8.5.1.3 UDC Resume (RSM)

When the UDC is in a suspend state, this bit can be written to force the UDC into a non-idle state (K state) for 3 ms to perform a remote wake-up operation. If the host PC does not start a wake-up sequence in 3 ms, the UDC returns to the suspend mode.

This bit is a trigger bit for the UDC and is automatically cleared.

8.5.1.4 Resume Interrupt Request (RESIR)

The resume interrupt request bit is set if the SRM bit in the UDC control register is cleared, the UDC is currently in the suspended state, and the USB is driven with resume signalling.

8.5.1.5 Suspend Interrupt Request (SUSIR)

The suspend interrupt request register is set when the USB remains idle for more than 6 ms. The SUSIR bit retains state so software can determine that the USB is idle. If SRM is 0, SUSIR being set will not generate an interrupt but status continues to be updated.

8.5.1.6 Suspend/Resume Interrupt Mask (SRM)

The suspend/resume interrupt mask (SRM) masks or enables the suspend interrupt request to the interrupt controller. When SRM is 1, the interrupt is masked and the setting of SUSIR will not generate an interrupt. When SRM is 0, the setting of SUSIR generates an interrupt when the USB is idle for more than 6 ms.

Programming SRM does not affect the state of SUSIR.

8.5.1.7 Reset Interrupt Request (RSTIR)

The reset interrupt request register is set when the host issues a reset. When the host issues a reset, the entire UDC is reset. The RSTIR bit retains its state so software can determine that the design was reset. If REM is 0, RSTIR being set does not generate an interrupt but status continues to be updated.

8.5.1.8 Reset Interrupt Mask (REM)

The reset interrupt mask (REM) masks or enables the reset interrupt request to the interrupt controller. When REM is 1, the interrupt is masked and the setting of RSTIR does not generate an interrupt. When REM is 0, the RSTIR setting generates an interrupt when the USB host controller issues an UDC reset.

Programming REM does not affect the state of RSTIR.

The UDE bit is cleared to 0, which disables the UDC following a Intel XScale processor reset. Writes to reserved bits are ignored and reads return zeros.

Register		UDCCR
Bits	Name	Description
31:8		Reserved for future use
7	REM	Reset interrupt mask (read/write). 0 = Reset interrupt enabled. 1 = Reset interrupt disabled.
6	RSTIR	Reset interrupt request (read/write 1 to clear). 1 = UDC was reset by the host.
5	SRM	Suspend/resume interrupt mask (read/write). 0 = Suspend/resume interrupt enabled. 1 = Suspend/resume interrupt disabled.
4	SUSIR	Suspend interrupt request (read/write 1 to clear). 1 = UDC received suspend signalling from the host.
3	RESIR	Resume interrupt request (read/write 1 to clear). 1 = UDC received resume signalling from the host.
2	RSM	Device Resume (read/write 1 to set). 0 = Maintain UDC suspend state 1 = Force UDC out of suspend
1	UDA	UDC active (read-only). 0 = UDC currently inactive. 1 = UDC currently active.
0	UDE	UDD enable (read/write). 0 = UDD disable 1 = UDD enabled, UDC+ and UDC- used for USB serial transmission/reception.

(UDCCS0)

293



8.5.2.1 OUT Packet Ready (OPR)

The OUT packet ready bit is set by the UDC when it receives a valid OUT packet to endpoint 0. When this bit is set, the USIR0[IR0] bit will be set in the UDC status/interrupt register if endpoint zero interrupts are enabled. This bit is cleared by writing a 1. The UDC is not allowed to enter the data phase of a transaction until this bit is cleared.

8.5.2.2 IN Packet Ready (IPR)

The IN packet ready bit is set by the Intel XScale processor if less than max_packet bytes (16) have been written to the endpoint 0 FIFO to be transmitted. The Intel XScale processor must not set this bit if a max_packet is to be transmitted. The UDC clears this bit when the packet has been successfully transmitted, the UDCCS0[FTF] bit has been set, or a control OUT is received. When this bit is cleared due to a successful IN transmission or the reception of a control OUT, the USIR0[IR0] bit in the UDC interrupt register is set if the endpoint 0 interrupt is enabled via UICR0[IM0]. The Intel XScale processor is not able to clear UDCCS0[IPR] and always reads back a 0.

When software enables the status stage for Vendor/Class commands and control data commands such as GET_DESCRIPTOR, GET_CONFIGURATION, GET_INTERFACE, GET_STATUS, and SET_DESCRIPTOR, software must also set IPR. The data in the Transmit FIFO must be transmitted and the interrupt must be processed before the IPR is set for the status stage.

The status stage for all other USB Standard Commands that do not have a data stage, such as SET_ADDRESS, SET_CONFIGURATION, SET_INTERFACE, SET_FEATURE, and CLEAR_FEATURE, is handled by the UDC and the software must not set IPR.

8.5.2.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers the reset of the endpoint 0 transmit FIFO. It is set when software writing a 1 or when the UDC receives an OUT packet from the host on endpoint 0. This bit always reads back a 0 value.

8.5.2.4 Device Remote Wake-Up Feature (DRWF)

The host indicates the state of the device remote wake-up feature by sending a Set Feature command or a Clear Function command. The UDC decodes the command sent by the host and sets this bit to a 1 if the feature is enabled and a 0 if the feature is disabled.

This bit is read-only.

8.5.2.5 Sent Stall (SST)

The sent stall bit is set by the UDC when FST successfully forces a software-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host when a STALL handshake is returned automatically. In this event, there is no intervention by the Intel XScale processor and the UDC clears the STALL status before the host sends the next SETUP command.

When the UDC sets this bit, the transmit FIFO is flushed. The Intel XScale processor writes a 1 to this bit to clear it.



Register		UDCCS0 (Sheet 2 of 2)
Bits	Name	Description
2	FTF	Flush Tx FIFO (always read 0/write 1 to set) 1 = Flush the contents of Tx FIFO.
1	IPR	IN packet ready (always read 0/write 1 to set). 1 = IN packet ready.
0	OPR	OUT packet ready (read/write 1 to clear) 1 = OUT packet ready.

8.5.3 UDC Endpoint 1 Control/Status Register

(UDCCS1)

The UDC Endpoint 1 control status register contains 6 bits that are used to operate endpoint 1, a Bulk IN endpoint.

8.5.3.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO. A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS1[TSP].

8.5.3.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR1 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled. This bit can be used to validate the other status/error bits in the endpoint 1 control/status register.

The UDCCS1[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR1 bit in the appropriate UDC status/interrupt register, but the IR1 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS1[TSP].

8.5.3.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.3.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS1[TUR] does not generate an interrupt and is for status only. UDCCS1[TUR] is cleared by writing a 1 to it.



Register		UDCCS1
Bits	Name	Description
31:8	(Reserved)	
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data 1 = Transmit FIFO has room for at least 1 complete data packet

8.5.4 UDC Endpoint 2 Control/Status Register

(UDCCS2)

The UDC Endpoint 2 Control/Status Register contains seven bits that are used to operate endpoint 2, a Bulk OUT endpoint.

8.5.4.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 64 bytes, a short packet, or a zero packet. This bit is not cleared until all data has been read from both buffers.

8.5.4.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR2 bit in the appropriate UDC status/interrupt register is set, if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 2 Control/Status Register. The UDCCS2[RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

8.5.4.3 Bit 2 Reserved

Bit 2 is reserved for future use.

8.5.4.4 Bit 2 Reserved

Bit 3 is reserved for future use.



8.5.4.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

Any valid data in the FIFO remains valid and the software must unload it. The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS2[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS2[FST] bit must be set again.

The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.4.6 Force Stall (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS2[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS2[FST] bit is set. The UDCCS2[FST] bit is automatically cleared when the UDCCS2[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS2[FTF] bit.

8.5.4.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO.

This bit must be polled when the UDCCS2[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale processor did not read. The receive FIFO must continue to be read until this bit clears or data will be lost.

8.5.4.8 Receive Short Packet (RSP)

The UDC uses the receive short packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS2[RSP] is a 1 and UDCCS2[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register. UDCCS2[RSP] is cleared when the next OUT packet is received.



Register Name:		UDCCS2																							
Hex Offset Address:		0 x C800 B018								Reset Hex Value:		0 x 00000000													
Register Description:		Universal Serial Bus Device Controller Endpoint 2 Control and Status Register																							
Access: Read/Write																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)															RSP	RNE	FST	SST	DME	(Rsvd)	RPC	RFS			
X															0	0	0	0	0	0	0	0	0		
Resets (Above)																									

Register		UDCCS2
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has 1 or more data packets.

8.5.5 UDC Endpoint 3 Control/Status Register

(UDCCS3)

The UDC Endpoint 3 control status register contains four bits that are used to operate Endpoint 3, an Isochronous IN endpoint.

8.5.5.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if one or fewer data packets remain in the transmit FIFO. UDCCS3[TFS] is cleared when two complete data packets are in the FIFO.

A complete packet of data is signified by loading 256 bytes or by setting UDCCS3[TSP].



8.5.5.2 Transmit Packet Complete (TPC)

The UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR3 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the endpoint 3 control/status register. The UDCCS3[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR3 bit in the appropriate UDC status/interrupt register, but the IR3 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS3[TSP].

8.5.5.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.5.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS3[TUR] generates an interrupt.

UDCCS3[TUR] is cleared by writing a 1 to it.

8.5.5.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.5.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.5.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.5.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.



Register Name:		UDCCS3																							
Hex Offset Address:		0 x C800 B01C								Reset Hex Value:		0 x 00000001													
Register Description:		Register Description: Universal Serial Bus Device Controller Endpoint 3 Control and Status Register																							
Access: Read/Write																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)															TSP	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS			
															0	0	0	0	0	0	0	1			
Resets (Above)																									



8.5.6.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR4 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled. This bit can be used to validate the other status/error bits in the Endpoint 4 control/status register.

The UDCCS4[RPC] bit is cleared by writing a 1 to it.

8.5.6.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR4 in the appropriate UDC status/interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them.

This bit is cleared by writing a 1 to it.

8.5.6.4 Bit 3 Reserved

Bit 3 is reserved for future use.

8.5.6.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.6.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.6.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS4[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

8.5.6.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet.

This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer. If UDCCS4[RSP] is a 1 and UDCCS4[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register.

UDCCS4[RSP] clears when the next OUT packet is received.



Register Name:		UDCCS4																							
Hex Offset Address:		0x C800B020								Reset Hex Value:		0x00000000													
Register Description:		Universal Serial Bus Device Controller Endpoint 4 Control and Status Register																							
Access: Read/Write																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)																	RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS	
X																	0	0	0	0	0	0	0	0	
Resets (Above)																									

Register		UDCCS4
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3		(Reserved)
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

8.5.7 UDC Endpoint 5 Control /Status Register

(UDCCS5)

The UDC Endpoint 5 Control Status Register contains six bits that are used to operate Endpoint 5, an Interrupt IN endpoint.

8.5.7.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS5[TSP] is not set.



8.5.7.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR5 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 5 Control/Status Register.

The UDCCS5[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR5 bit in the appropriate UDC status/interrupt register, but the IR5 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS5[TSP].

8.5.7.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO.

The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE. The bit's read value is 0.

8.5.7.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host.

UDCCS5[TUR] does not generate an interrupt and is for status only. UDCCS5[TUR] is cleared by writing a 1 to it.

8.5.7.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS5[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS5[FST] bit must be set again.

The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.7.6 Force STALL (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS5[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS5[FST] bit is set. The UDCCS5[FST] bit is automatically cleared when the UDCCS5[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS5[FTF] bit.



8.5.7.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.7.8 Transmit Short Packet (TSP)

Software uses the transmit short to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit.

Software must not set this bit if a packet of 8 bytes is to be transmitted. When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:		UDCCS5																															
Hex Offset Address:		0x C800B024								Reset Hex Value:		0x00000001																					
Register Description:		Universal Serial Bus Device Controller Endpoint 5 Control and Status Register																															
Access: Read/Write																																	
Bits																																	
31																		8	7	6	5	4	3	2	1	0							
(Reserved)																			TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS							
X																			0	0	0	0	0	0	0	0	1						
Resets (Above)																																	

Register		UDCCS5
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.



8.5.8 UDC Endpoint 6 Control/Status Register

(UDCCS6)

The UDC Endpoint 6 Control Status Register contains six bits that are used to operate Endpoint 6, a Bulk IN endpoint.

8.5.8.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO.

A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS6[TSP].

8.5.8.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR6 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled. This bit can be used to validate the other status/error bits in the Endpoint 6 Control/Status Register.

The UDCCS6[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR6 bit in the appropriate UDC status/interrupt register, but the IR6 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS6[TSP].

8.5.8.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO.

The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE. The bit's read value is 0.

8.5.8.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS6[TUR] does not generate an interrupt and is for status only.

UDCCS6[TUR] is cleared by writing a 1 to it.

8.5.8.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS6[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS6[FST] bit must be set again. The Intel XScale processor writes a 1 to the sent stall bit to clear it.



8.5.8.6 Force STALL (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS6[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS6[FST] bit is set. The UDCCS6[FST] bit is automatically cleared when the UDCCS6[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS6[FTF] bit.

8.5.8.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.8.8 Transmit Short Packet (TSP)

The software uses the transmit short packet bit to indicate that the last byte of a data transfer to the FIFO has occurred. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit if a 64-byte packet is to be transmitted. When the data packet is successful transmitted, the UDC clears this bit.

Register Name:		UDCCS6																															
Hex Offset Address:		0 x C800B028								Reset Hex Value:		0 x 00000001																					
Register Description:		Universal Serial Bus Device Controller Endpoint 6Control and Status Register																															
Access: Read/Write																																	
Bits																																	
31																		8	7	6	5	4	3	2	1	0							
(Reserved)																		TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS								
X																		0	0	0	0	0	0	0	0	1							
Resets (Above)																																	

Register		UDCCS6 (Sheet 1 of 2)
Bits	Name	Description
31:8		(Reserved)
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.



Register		UDCCS6 (Sheet 2 of 2)
Bits	Name	Description
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

8.5.9 UDC Endpoint 7 Control/Status Register

(UDCCS7)

The UDC Endpoint 7 Control/Status Register contains seven bits that are used to operate Endpoint 7, a Bulk OUT endpoint.

8.5.9.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 64 bytes, a short packet, or a zero packet.

This bit is not cleared until all data has been read from both buffers.

8.5.9.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR7 bit in the appropriate UDC status/interrupt register is set — if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 7 Control/Status Register. The UDCCS7 [RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

8.5.9.3 Bit 2 Reserved

Bit 2 is reserved for future use.

8.5.9.4 Bit 3 Reserved

Bit 3 is reserved for future use.

8.5.9.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.



Any valid data in the FIFO remains valid and the software must unload it. The end-point operation continues normally and does not send another STALL condition, even if the UDCCS7[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS7[FST] bit must be set again.

The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.9.6 Force Stall (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS7[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS7[FST] bit is set. The UDCCS7[FST] bit is automatically cleared when the UDCCS7[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS7[FTF] bit.

8.5.9.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO. This bit must be polled when the UDCCS7[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

8.5.9.8 Receive Short Packet (RSP)

The UDC uses the receive-short-packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS7[RSP] is a 1 and UDCCS7[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register.

UDCCS7[RSP] is cleared when the next OUT packet is received.

Register Name:		UDCCS7																								
Hex Offset Address:		0 x C800 B02C								Reset Hex Value:		0 x 00000000														
Register Description:		Universal Serial Bus Device Controller Endpoint 7Control and Status Register																								
Access: Read/Write																										
Bits																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																			RSP	RNE	FST	SST	(Rsvd)	(Rsvd)	RPC	RFS
X																			0	0	0	0	0	0	0	0
Resets (Above)																										



Register		UDCCS7
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

8.5.10 UDC Endpoint 8 Control /Status Register

(UDCCS8)

The UDC Endpoint 8 Control Status Register contains four bits that are used to operate Endpoint 8, an isochronous IN endpoint.

8.5.10.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is be set if one or fewer data packets remain in the transmit FIFO. UDCCS8[TFS] is cleared when two complete data packets are in the FIFO. A complete packet of data is signified by loading 256 bytes or by setting UDCCS8[TSP].

8.5.10.2 Transmit Packet Complete (TPC)

The the UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR8 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 8 Control/Status Register. The UDCCS8[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR8 bit in the appropriate UDC status/interrupt register, but the IR8 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS8[TSP].



8.5.10.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.10.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS8[TUR] generates an interrupt.

UDCCS8[TUR] is cleared by writing a 1 to it.

8.5.10.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.10.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.10.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.10.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.

Register Name:		UDCCS8																																
Hex Offset Address:		0 x C800 B030								Reset Hex Value:		0 x 00000001																						
Register Description:		Register Description: Universal Serial Bus Device Controller Endpoint 8Control and Status Register																																
Access: Read/Write																																		
Bits																																		
31																										8	7	6	5	4	3	2	1	0
(Reserved)																				TSP	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS							
																				0	0	0	0	0	0	0	0	1						
Resets (Above)																																		



Register		UDCCS8
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

8.5.11 UDC Endpoint 9 Control/Status Register

(UDCCS9)

The UDC Endpoint 9 Control/Status Register contains six bits that are used to operate Endpoint 9, an isochronous OUT endpoint.

8.5.11.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 256 bytes, a short packet, or a zero packet. UDCCS9[RFS] is not cleared until all data is read from both buffers.

8.5.11.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR9 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 9 Control/Status Register. The UDCCS9[RPC] bit is cleared by writing a 1 to it.

8.5.11.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR9 in the appropriate UDC status/interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them. This bit is cleared by writing a 1 to it.

8.5.11.4 Bit 3 Reserved

Bit 3 is reserved for future use.



8.5.11.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.11.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.11.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS9[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

8.5.11.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS9[RSP] is a one and UDCCS9[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register.

UDCCS9[RSP] clears when the next OUT packet is received.

Register Name:		UDCCS9																							
Hex Offset Address:		0x C800B034								Reset Hex Value:		0x00000000													
Register Description:		Universal Serial Bus Device Controller Endpoint 9Control and Status Register																							
Access: Read/Write																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)																RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS		
X																0	0	0	0	0	0	0	0		
Resets (Above)																									

Register		UDCCS9 (Sheet 1 of 2)
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.



Register		UDCCS9 (Sheet 2 of 2)
Bits	Name	Description
3		(Reserved)
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has one or more data packets.

8.5.12 UDC Endpoint 10 Control/Status Register

(UDCCS10)

The UDC Endpoint 10 Control Status Register contains six bits that are used to operate Endpoint 10, an Interrupt IN endpoint.

8.5.12.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS10[TSP] is not set.

8.5.12.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is be set by the UDC when an entire packet is sent to the host. When this bit is set, the IR10 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 10 Control/Status Register. The UDCCS10[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR10 bit in the appropriate UDC status/interrupt register, but the IR10 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS10[TSP].

8.5.12.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.12.4 Transmit Underrun (TUR)

The transmit underrun bit is be set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS10[TUR] does not generate an interrupt and is for status only. UDCCS10[TUR] is cleared by writing a 1 to it.



8.5.12.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS10[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS10[FST] bit must be set again. The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.12.6 Force STALL (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS10[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS10[FST] bit is set. The UDCCS10[FST] bit is automatically cleared when the UDCCS10[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS10[FTF] bit.

8.5.12.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.12.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit if a packet of 8 bytes is to be transmitted. When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:		UDCCS10																										
Hex Offset Address:		0x C800B038								Reset Hex Value:		0x00000001																
Register Description:		Universal Serial Bus Device Controller Endpoint 10Control and Status Register																										
Access: Read/Write																												
Bits																												
31																				8	7	6	5	4	3	2	1	0
(Reserved)																				TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS	
X																				0	0	0	0	0	0	0	0	1
Resets (Above)																												



Register		UDCCS10
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.

8.5.13 UDC Endpoint 11 Control / Status Register

(UDCCS11)

The UDC Endpoint 11 Control Status Register contains six bits that are used to operate Endpoint 11, a Bulk IN endpoint.

8.5.13.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is active if one or fewer data packets remain in the transmit FIFO. TFS is cleared when two complete packets of data remain in the FIFO. A complete packet of data is signified by loading 64 bytes of data or by setting UDCCS11[TSP].

8.5.13.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR11 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 11 Control/Status Register. The UDCCS11[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR11 bit in the appropriate UDC status/interrupt register, but the IR11 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS11[TSP].



8.5.13.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.13.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS11[TUR] does not generate an interrupt and is for status only.

UDCCS11[TUR] is cleared by writing a 1 to it.

8.5.13.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS11[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS11[FST] bit must be set again.

The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.13.6 Force STALL (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS11[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS11[FST] bit is set. The UDCCS11[FST] bit is automatically cleared when the UDCCS11[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS11[FTF] bit.

8.5.13.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.13.8 Transmit Short Packet (TSP)

The software uses the transmit short packet bit to indicate that the last byte of a data transfer to the FIFO has occurred. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit if a 64-byte packet is to be transmitted. When the data packet is successful transmitted, the UDC clears this bit.

Register		UDCCS11
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least 1 complete data packet.

(UDCCS12)

8.5.14.1 Receive FIFO Service (RFS)

This bit is not cleared until all data has been read from both buffers.



8.5.14.2 Receive Packet Complete (RPC)

The receive packet complete bit is set by the UDC when an OUT packet is received. When this bit is set, the IR12 bit in the appropriate UDC status/interrupt register is set, if receive interrupts are enabled.

This bit can be used to validate the other status/error bits in the endpoint 12 control/status register. The UDCCS12[RPC] bit is cleared by writing a 1 to it. The UDC issues NAK handshakes to all OUT tokens while this bit is set and both buffers have unread data.

8.5.14.3 Bit 2 Reserved

Bit 2 is reserved for future use.

8.5.14.4 Bit 3 Reserved

Bit 3 is reserved for future use.

8.5.14.5 Sent Stall (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

Any valid data in the FIFO remains valid and the software must unload it. The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS12[SST] bit is set.

To allow the software to continue to send the STALL condition on the USB bus, the UDCCS12[FST] bit must be set again. The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.14.6 Force Stall (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all OUT tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS12[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS12[FST] bit is set. The UDCCS12[FST] bit is automatically cleared when the UDCCS12[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS12[FTF] bit.

8.5.14.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that unread data remains in the receive FIFO. This bit must be polled when the UDCCS12[RPC] bit is set to determine if there is any data in the FIFO that the Intel XScale processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.



8.5.14.8 Receive Short Packet (RSP)

The UDC uses the receive short packet bit to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS12[RSP] is a 1 and UDCCS12[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register. UDCCS12[RSP] is cleared when the next OUT packet is received.

Register Name:		UDCCS12																								
Hex Offset Address:		0 x C800 B040								Reset Hex Value:		0 x 00000000														
Register Description:		Universal Serial Bus Device Controller Endpoint 12 Control and Status Register																								
Access: Read/Write																										
Bits																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																			RSP	RNE	FST	SST	(Rsvd)	(Rsvd)	RPC	RFS
X																			0	0	0	0	0	0	0	0
Resets (Above)																										

Register		UDCCS12
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5	FST	Force stall (read/write). 1 = Issue STALL handshakes to OUT tokens.
4	SST	Sent stall (read/write 1 to clear). 1 = STALL handshake was sent.
3		(Reserved)
2		(Reserved). Always reads zero.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than 1 data packet. 1 = Receive FIFO has 1 or more data packets.

8.5.15 UDC Endpoint 13 Control/Status Register

(UDCCS13)

The UDC Endpoint 13 Control Status Register contains four bits that are used to operate Endpoint 13, an Isochronous IN endpoint.



8.5.15.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if one or fewer data packets remain in the transmit FIFO. UDCCS13[TFS] is cleared when two complete data packets are in the FIFO. A complete packet of data is signified by loading 256 bytes or by setting UDCCS13[TSP].

8.5.15.2 Transmit Packet Complete (TPC)

The UDC sets transmit packet complete bit when an entire packet is sent to the host. When this bit is set, the IR13 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 13 Control/Status Register. The UDCCS13[TPC] bit gets cleared by writing a 1 to it. This clears the interrupt source for the IR13 bit in the appropriate UDC status/interrupt register, but the IR13 bit must also be cleared.

Setting this bit does not prevent the UDC from transmitting the next buffer. The UDC issues NAK handshakes to all IN tokens if this bit is set and neither buffer has been triggered by writing 64 bytes or setting UDCCS13[TSP].

8.5.15.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is zero.

8.5.15.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, UDCCS13[TUR] generates an interrupt.

UDCCS13[TUR] is cleared by writing a 1 to it.

8.5.15.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.15.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.15.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.15.8 Transmit Short Packet (TSP)

Software uses the transmit short packet to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software should always check TSP after loading a packet to determine if more data can be loaded.

Software must not set this bit, if a packet of 256 bytes is to be transmitted. When the data packet is successfully transmitted, this bit is cleared by the UDC.



Register Name:		UDCCS13																								
Hex Offset Address:		0 x C800 B044								Reset Hex Value:		0 x 00000001														
Register Description:		Register Description: Universal Serial Bus Device Controller Endpoint 13 Control and Status Register																								
Access: Read/Write																										
Bits																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																		TSp	(Rsvd)	(Rsvd)	(Rsvd)	TUR	FTF	TPC	TFS	
																		0	0	0	0	0	0	0	1	
Resets (Above)																										

Register		UDCCS13
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for at least one complete data packet.

8.5.16 UDC Endpoint 14 Control / Status Register

(UDCCS14)

The UDC Endpoint 14 Control/Status Register contains six bits that are used to operate Endpoint 14, an Isochronous OUT endpoint.

8.5.16.1 Receive FIFO Service (RFS)

The receive FIFO service bit is set if the receive FIFO has one complete data packet in it and the packet has been error checked by the UDC. A complete packet may be 256 bytes, a short packet, or a zero packet.

UDCCS14[RFS] is not cleared until all data is read from both buffers.



8.5.16.2 Receive Packet Complete (RPC)

The receive packet complete bit gets set by the UDC when an OUT packet is received. When this bit is set, the IR14 bit in the appropriate UDC status/interrupt register is set if receive interrupts are enabled. This bit can be used to validate the other status/error bits in the Endpoint 14 control/status register.

The UDCCS14[RPC] bit is cleared by writing a 1 to it.

8.5.16.3 Receive Overflow (ROF)

The receive overflow bit generates an interrupt on IR14 in the appropriate UDC status/interrupt register to alert the software that Isochronous data packets are being dropped because neither FIFO buffer has room for them.

This bit is cleared by writing a 1 to it.

8.5.16.4 Bit 3 Reserved

Bit 3 is reserved for future use.

8.5.16.5 Bit 4 Reserved

Bit 4 is reserved for future use.

8.5.16.6 Bit 5 Reserved

Bit 5 is reserved for future use.

8.5.16.7 Receive FIFO Not Empty (RNE)

The receive FIFO not empty bit indicates that the receive FIFO has unread data in it. When the UDCCS14[RPC] bit is set, this bit must be read to determine if there is any data in the FIFO that Intel XScale processor did not read.

The receive FIFO must continue to be read until this bit clears or data will be lost.

8.5.16.8 Receive Short Packet (RSP)

The receive short packet bit is used by the UDC to indicate that the received OUT packet in the active buffer currently being read is a short packet or zero-sized packet. This bit is updated by the UDC after the last byte is read from the active buffer and reflects the status of the new active buffer.

If UDCCS14[RSP] is a 1 and UDCCS14[RNE] is a 0, it indicates a zero-length packet. If a zero-length packet is present, the Intel XScale processor must not read the data register.

UDCCS14[RSP] clears when the next OUT packet is received.



Register Name:		UDCCS14																									
Hex Offset Address:		0x C800B048								Reset Hex Value:		0x00000000															
Register Description:		Universal Serial Bus Device Controller Endpoint 14 Control and Status Register																									
Access: Read/Write																											
Bits																											
31																			8	7	6	5	4	3	2	1	0
(Reserved)																			RSP	RNE	(Rsvd)	(Rsvd)	(Rsvd)	ROF	RPC	RFS	
X																			0	0	0	0	0	0	0	0	
Resets (Above)																											

Register		UDCCS14
Bits	Name	Description
31:8		Reserved for future use.
7	RSP	Receive short packet (read only). 1 = Short packet received and ready for reading.
6	RNE	Receive FIFO not empty (read-only). 0 = Receive FIFO empty. 1 = Receive FIFO not empty.
5		(Reserved). Always reads 0.
4		(Reserved). Always reads 0.
3		(Reserved)
2	ROF	Receive overflow (read/write 1 to clear). 1 = Isochronous data packets are being dropped from the host because the receiver is full.
1	RPC	Receive packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Receive packet has been received and error/status bits are valid.
0	RFS	Receive FIFO service (read-only). 0 = Receive FIFO has less than one data packet. 1 = Receive FIFO has one or more data packets.

8.5.17 UDC Endpoint 15 Control /Status Register

(UDCCS15)

The UDC Endpoint 15 Control Status Register contains six bits that are used to operate endpoint 15, an Interrupt IN endpoint.

8.5.17.1 Transmit FIFO Service (TFS)

The transmit FIFO service bit is set if the FIFO does not contain any data bytes and UDCCS15[TSP] is not set.



8.5.17.2 Transmit Packet Complete (TPC)

The transmit packet complete bit is set by the UDC when an entire packet is sent to the host. When this bit is set, the IR15 bit in the appropriate UDC status/interrupt register is set if transmit interrupts are enabled.

This bit can be used to validate the other status/error bits in the Endpoint 15 Control/Status Register.

The UDCCS15[TPC] bit is cleared by writing a 1 to it. This clears the interrupt source for the IR15 bit in the appropriate UDC status/interrupt register, but the IR15 bit must also be cleared.

The UDC issues NAK handshakes to all IN tokens if this bit is set and the buffer is not triggered by writing 8 bytes or setting UDCCS15[TSP].

8.5.17.3 Flush Tx FIFO (FTF)

The Flush Tx FIFO bit triggers a reset for the endpoint's transmit FIFO. The Flush Tx FIFO bit is set when software writes a 1 to it or when the host performs a SET_CONFIGURATION or SET_INTERFACE.

The bit's read value is 0.

8.5.17.4 Transmit Underrun (TUR)

The transmit underrun bit is set if the transmit FIFO experiences an underrun. When the UDC experiences an underrun, NAK handshakes are sent to the host. UDCCS15[TUR] does not generate an interrupt and is for status only.

UDCCS15[TUR] is cleared by writing a 1 to it.

8.5.17.5 Sent STALL (SST)

The sent stall bit is set by the UDC in response to FST successfully forcing a user-induced STALL on the USB bus. This bit is not set if the UDC detects a protocol violation from the host PC when a STALL handshake is returned automatically. In either event, the Intel XScale processor does not intervene and the UDC clears the STALL status when the host sends a CLEAR_FEATURE command.

The endpoint operation continues normally and does not send another STALL condition, even if the UDCCS15[SST] bit is set. To allow the software to continue to send the STALL condition on the USB bus, the UDCCS15[FST] bit must be set again.

The Intel XScale processor writes a 1 to the sent stall bit to clear it.

8.5.17.6 Force STALL (FST)

The Intel XScale processor can set the force stall bit to force the UDC to issue a STALL handshake to all IN tokens. STALL handshakes continue to be sent until the Intel XScale processor clears this bit by sending a Clear Feature command.

The UDCCS15[SST] bit is set when the STALL state is actually entered, but this may be delayed if the UDC is active when the UDCCS15[FST] bit is set. The UDCCS15[FST] bit is automatically cleared when the UDCCS15[SST] bit is set.

To ensure that no data is transmitted after the Clear Feature command is sent and the host resumes IN requests, software must clear the transmit FIFO by setting the UDCCS15[FTF] bit.



8.5.17.7 Bit 6 Reserved

Bit 6 is reserved for future use.

8.5.17.8 Transmit Short Packet (TSP)

Software uses the transmit short to indicate that the last byte of a data transfer has been sent to the FIFO. This indicates to the UDC that a short packet or zero-sized packet is ready to transmit. Software must not set this bit if a packet of 8 bytes is to be transmitted.

When the data packet is successfully transmitted, the UDC clears this bit.

Register Name:		UDCCS15																							
Hex Offset Address:		0x C800B04C								Reset Hex Value:		0x00000001													
Register Description:		Universal Serial Bus Device Controller Endpoint 15 Control and Status Register																							
Access: Read/Write																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)																	TSP	(Rsvd)	FST	SST	TUR	FTF	TPC	TFS	
X																	0	0	0	0	0	0	0	1	
Resets (Above)																									

Register		UDCCS15
Bits	Name	Description
31:8		Reserved for future use.
7	TSP	Transmit short packet (read/write 1 to set). 1 = Short packet ready for transmission.
6		(Reserved). Always reads 0.
5	FST	Force STALL (read/write). 1 = Issue STALL handshakes to IN tokens.
4	SST	Sent STALL (read/write 1 to clear). 1 = STALL handshake was sent.
3	TUR	Transmit FIFO underrun (read/write 1 to clear). 1 = Transmit FIFO experienced an underrun.
2	FTF	Flush Tx FIFO (always read 0/ write a 1 to set). 1 = Flush Contents of TX FIFO.
1	TPC	Transmit packet complete (read/write 1 to clear). 0 = Error/status bits invalid. 1 = Transmit packet has been sent and error/status bits are valid.
0	TFS	Transmit FIFO service (read-only). 0 = Transmit FIFO has no room for new data. 1 = Transmit FIFO has room for 1 complete data packet.



8.5.18 UDC Interrupt Control Register 0

(UICR0)

The UICR0 contains eight control bits to enable/disable interrupt service requests from data endpoints 0 - 7. All of the UICR0 bits are reset to a 1 so interrupts are not generated on initial system reset.

8.5.18.1 Interrupt Mask Endpoint x (IMx), where x is 0 through 7

The UICR0[IMx] bit is used to mask or enable the corresponding endpoint interrupt request, USIR0[IRx]. When the mask bit is set, the interrupt is masked and the corresponding bit in the USIR0 register is not allowed to be set.

When the mask bit is cleared and an interruptible condition occurs in the endpoint, the appropriate interrupt bit is set. Programming the mask bit to a 1 does not affect the current state of the interrupt bit. It only blocks future 0-to-1 transitions of the interrupt bit.

Register Name:		UICR0																															
Hex Offset Address:		0 x C800B050								Reset Hex Value:		0x000000FF																					
Register Description:		Universal Serial Bus Device Controller Interrupt Control Register 0																															
Access: Read/Write and Read-Only																																	
Bits																																	
31																			8	7	6	5	4	3	2	1	0						
(Reserved)																			IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0							
X																			1	1	1	1	1	1	1	1							
Resets (Above)																																	

Register		UICR0 (Sheet 1 of 2)
Bits	Name	Description
31:8		Reserved for future use.
7	IM7	Interrupt Mask for Endpoint 7. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
6	IM6	Interrupt Mask for Endpoint 6. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
5	IM5	Interrupt mask for Endpoint 5. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
4	IM4	Interrupt mask for Endpoint 4. 0 = Receive Interrupt enabled. 1 = Receive Interrupt disabled.
3	IM3	Interrupt mask for Endpoint 3. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.



Register		UICR0 (Sheet 2 of 2)
Bits	Name	Description
2	IM2	Interrupt Mask for Endpoint 2. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
1	IM1	Interrupt Mask for Endpoint 1. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
0	IM0	Interrupt mask for endpoint 0. 0 = Endpoint zero interrupt enabled. 1 = Endpoint zero interrupt disabled.

8.5.19 UDC Interrupt Control Register 1

(UICR1)

The UICR1 contains eight control bits to enable/disable interrupt service requests from Endpoints 8 through 15. The UICR1 bits are reset to 1 so interrupts are not generated on initial system reset.

8.5.19.1 Interrupt Mask Endpoint x (IMx), where x is 8 through 15

The UICR1[IMx] bit is used to mask or enable the corresponding endpoint interrupt request, USIR1[IRx]. When the mask bit is set, the interrupt is masked and the corresponding bit in the USIR1 register is not allowed to be set.

When the mask bit is cleared and an interruptible condition occurs in the endpoint, the appropriate interrupt bit is set. Programming the mask bit to a 1 does not affect the current state of the interrupt bit. It only blocks future 0-to-1 transitions of the interrupt bit.

Register Name:		UICR1																								
Hex Offset Address:		0 x C800B054								Reset Hex Value:		0x000000FF														
Register Description:		Universal Serial Bus Device Controller Interrupt Control Register 1																								
Access: Read/Write																										
Bits																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																		IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	
																		1	1	1	1	1	1	1	1	
Resets (Above)																										



Register		UICR1
Bits	Name	Description
31:8		Reserved for future use.
7	IM15	Interrupt mask for Endpoint 15. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
6	IM14	Interrupt mask for Endpoint 14. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
5	IM13	Interrupt mask for Endpoint 13. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
4	IM12	Interrupt mask for Endpoint 12. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
3	IM11	Interrupt mask for Endpoint 11. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.
2	IM10	Interrupt mask for Endpoint 10. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
1	IM9	Interrupt mask for Endpoint 9. 0 = Receive interrupt enabled. 1 = Receive interrupt disabled.
0	IM8	Interrupt Mask for Endpoint 8. 0 = Transmit interrupt enabled. 1 = Transmit interrupt disabled.

8.5.20 UDC Status/ Interrupt Register 0

(UISR0)

The UDC status/interrupt registers (USIR0 and USIR1) contain bits that are used to generate the UDC's interrupt request. Each bit in the UDC status/interrupt registers is logically ORed together to produce one interrupt request.

When the ISR for the UDC is executed, it must read the UDC status/interrupt register to determine why the interrupt occurred.

The bits in USIR0 and USIR1 are controlled by a mask bit in the UDC Interrupt Control Register (UICR0/1). The mask bits, when set, prevent a status bit in the USIRx from being set. If the mask bit for a particular status bit is cleared and an interruptible condition occurs, the status bit is set.

To clear status bits, the Intel XScale processor must write a 1 to the position to be cleared. The interrupt request for the UDC remains active as long as the value of the USIRx is non-zero.

8.5.20.1 Endpoint 0 Interrupt Request (IR0)

The endpoint 0 interrupt request is set if the IM0 bit in the UDC control register is cleared and, in the UDC endpoint 0 control/status register, the OUT packet ready bit is set, the IN packet ready bit is cleared, or the sent STALL bit is set. The IR0 bit is cleared by writing a 1 to it.



8.5.20.2 Endpoint 1 Interrupt Request (IR1)

The interrupt request bit is set if the IM1 bit in the UDC interrupt control register is cleared and the IN packet complete (TPC) in UDC endpoint 1 control/status register is set.

The IR1 bit is cleared by writing a 1 to it.

8.5.20.3 Endpoint 2 Interrupt Request (IR2)

The interrupt request bit is set if the IM2 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC Endpoint 2 Control/Status Register is set.

The IR2 bit is cleared by writing a 1 to it.

8.5.20.4 Endpoint 3 Interrupt Request (IR3)

The interrupt request bit is set if the IM3 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 3 Control/Status Register is set.

The IR3 bit is cleared by writing a 1 to it

8.5.20.5 Endpoint 4 Interrupt Request (IR4)

The interrupt request bit is set if the IM4 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 4 Control/Status Register or the Isochronous Error Endpoint 4 (IPE4) in the UFNHR are set.

The IR4 bit is cleared by writing a 1 to it.

8.5.20.6 Endpoint 5 Interrupt Request (IR5)

The interrupt request bit is set if the IM5 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC Endpoint 5 Control/Status Register is set.

The IR5 bit is cleared by writing a 1 to it.

8.5.20.7 Endpoint 6 Interrupt Request (IR6)

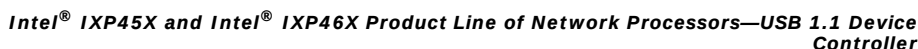
The interrupt request bit gets set if the IM6 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC Endpoint 6 control/status register gets set.

The IR6 bit is cleared by writing a 1 to it.

8.5.20.8 Endpoint 7 Interrupt Request (IR7)

The interrupt request bit is set if the IM7 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC Endpoint 7 Control/Status Register is set.

The IR7 bit is cleared by writing a 1 to it.



Register		USIR0
Bits	Name	Description
31:8		Reserved for future use.
7	IR7	Interrupt Request Endpoint 7 (read/write 1 to clear). 1 = Endpoint 7 needs service.
6	IR6	Interrupt Request Endpoint 6 (read/write 1 to clear). 1 = Endpoint 6 needs service.
5	IR5	Interrupt Request Endpoint 5 (read/write 1 to clear). 1 = Endpoint 5 needs service.
4	IR4	Interrupt Request Endpoint 4 (read/write 1 to clear). 1 = Endpoint 4 needs service.
3	IR3	Interrupt Request Endpoint 3 (read/write 1 to clear). 1 = Endpoint 3 needs service.
2	IR2	Interrupt Request Endpoint 2 (read/write 1 to clear). 1 = Endpoint 2 needs service.
1	IR1	Interrupt Request Endpoint 1 (read/write 1 to clear). 1 = Endpoint 1needs service.
0	IR0	Interrupt Request Endpoint 0 (read/write 1 to clear). 1 = Endpoint 0 needs service.

(USIR1)

The interrupt request bit is set if the IM8 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 8 Control/Status Register is set.

The interrupt request bit is set if the IM9 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 9 Control/Status Register or the Isochronous Error Endpoint 9 (IPE9) in the UFNHR are set.



The IR9 bit is cleared by writing a 1 to it.

8.5.21.3 Endpoint 10 Interrupt Request (IR10)

The interrupt request bit is set if the IM10 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or in UDC endpoint 10 control/status register is set.

The IR10 bit is cleared by writing a 1 to it.

8.5.21.4 Endpoint 11 Interrupt Request (IR11)

The interrupt request bit is set if the IM11 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) in UDC Endpoint 11 Control/Status Register is set.

The IR11 bit is cleared by writing a 1 to it.

8.5.21.5 Endpoint 12 Interrupt Request (IR12)

The interrupt request bit is set if the IM12 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready bit (RPC) in the UDC endpoint 12 control/status register is set.

The IR12 bit is cleared by writing a 1 to it.

8.5.21.6 Endpoint 13 Interrupt Request (IR13)

The interrupt request bit is set if the IM13 bit in the UDC Interrupt Control Register is cleared and the IN packet complete (TPC) or Transmit Underrun (TUR) in UDC Endpoint 13 Control/Status Register is set.

The IR13 bit is cleared by writing a 1 to it.

8.5.21.7 Endpoint 14 Interrupt Request (IR14)

The interrupt request bit is set if the IM14 bit in the UDC Interrupt Control Register is cleared and the OUT packet ready (RPC) or receiver overflow (ROF) in the UDC Endpoint 14 Control/Status Register or the Isochronous Error Endpoint 14 (IPE14) in the UFNHR are set.

The IR14 bit is cleared by writing a 1 to it.

8.5.21.8 Endpoint 15 Interrupt Request (IR15)

The interrupt request bit is set if the IM15 bit in the UDC interrupt control is set.

The IR15 bit is cleared by writing a 1 to it.



Register Name:		USIR1																							
Hex Offset Address:		0 x C800B05C								Reset Hex Value:		0x00000000													
Register Description:		Universal Serial Bus Device Controller Interrupt Status Register 1																							
Access: Read/Write and Read-Only																									
Bits																									
31																	8	7	6	5	4	3	2	1	0
(Reserved)																	IR15	IR14	IR13	IR12	IR11	IR10	IR9	IR8	
X																	0	0	0	0	0	0	0	0	
Resets (Above)																									

Register		USIR1
Bits	Name	Description
31:8		Reserved for future use.
7	IR15	Interrupt Request Endpoint 15 (read/write 1 to clear). 1 = Endpoint 15 needs service.
6	IR14	Interrupt Request Endpoint 14 (read/write 1 to clear). 1 = Endpoint 14 needs service.
5	IR13	Interrupt Request Endpoint 13 (read/write 1 to clear). 1 = Endpoint 13 needs service.
4	IR12	Interrupt Request Endpoint 12 (read/write 1 to clear). 1 = Endpoint 12 needs service.
3	IR11	Interrupt Request Endpoint 11 (read/write 1 to clear). 1 = Endpoint 11 needs service.
2	IR10	Interrupt Request Endpoint 10 (read/write 1 to clear). 1 = Endpoint 10 needs service.
1	IR9	Interrupt Request Endpoint 9 (read/write 1 to clear). 1 = Endpoint 9 needs service.
0	IR8	Interrupt Request Endpoint 8 (read/write 1 to clear). 1 = Endpoint 8 needs service.

8.5.22 UDC Frame Number High Register

(UFNHR)

The UDC Frame Number High Register holds the three most-significant bits of the frame number contained in the last received SOF packet, the isochronous OUT endpoint error status, and the SOF interrupt status/interrupt mask bit.

8.5.22.1 UDC Frame Number MSB (FNMSB)

The UFNHR[FNMSB] is the three most-significant bits of the 11-bit frame number contained in the last received SOF packet. The remaining bits are located in the UFNLR. This information is used for isochronous transfers.

These bits are updated every SOF.



8.5.22.2 Isochronous Packet Error Endpoint 4 (IPE4)

The isochronous packet error for Endpoint 4 is set if Endpoint 4 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of Endpoint 4.

To maintain synchronization, the software must monitor this bit when it services an SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

8.5.22.3 Isochronous Packet Error Endpoint 9 (IPE9)

The isochronous packet error for Endpoint 9 is set if Endpoint 9 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of Endpoint 9.

To maintain synchronization, software must monitor this bit when it services the SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

8.5.22.4 Isochronous Packet Error Endpoint 14 (IPE14)

The isochronous packet error for Endpoint 14 is set if Endpoint 14 is loaded with a data packet that is corrupted. This status bit is used in the interrupt generation of Endpoint 14.

To maintain synchronization, software must monitor this bit when it services the SOF interrupt and reads the frame number. This bit is not set if the token packet is corrupted or if the sync or PID fields of the data packet are corrupted.

8.5.22.5 Start of Frame Interrupt Mask (SIM)

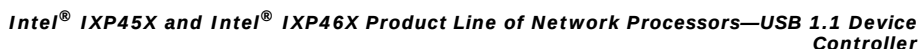
The UFNHR[SIM] bit is used to mask or enable the SOF interrupt request. When UFNHR[SIM]=1, the interrupt is masked and the SIR bit is not allowed to be set. When UFNHR[SIM]=0, the interrupt is enabled and when an interruptible condition occurs in the receiver, the UFNHR[SIR] bit is set.

Setting UFNHR[SIM] to a 1 does not affect the current state of UFNHR[SIR]. It only blocks future zero to one transitions of UFNHR[SIR].

8.5.22.6 Start of Frame Interrupt Request (SIR)

The interrupt request bit is set if the UFNHR[SIM] bit is cleared and an SOF packet is received.

The UFNHR[SIR] bit is cleared by writing a 1 to it.



Register		UFNHR
Bits	Name	Description
31:8		Reserved for future use.
7	SIR	SOF Interrupt Request (read/write 1 to clear). 1 = SOF has been received.
6	SIM	SOF interrupt mask. 0 = SOF interrupt enabled. 1 = SOF interrupt disabled.
5	IPE14	Isochronous Packet Error Endpoint 14 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
4	IPE9	Isochronous Packet Error Endpoint 9 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
3	IPE4	Isochronous Packet Error Endpoint 4 (read/write 1 to clear). 1 = Status indicator that data in the endpoint fifo is corrupted.
2:0	FNMSB	Frame Number MSB. Most-significant three bits of 11-bit frame number associated with last receive SOF. Reset to all zeros.

(UFNLR)

These bits are updated every SOF.



Register Name:		UFNLR																								
Hex Offset Address:		0 x C800B064								Reset Hex Value:		0x00000000														
Register Description:		Universal Serial Bus Device Frame Number Low Register																								
Access: Read-Only																										
Bits																										
31																		8	7							0
(Reserved)																		8-Bit Frame Number LSB								
X																		0	0	0	0	0	0	0	0	
Resets (Above)																										

Register		UFNLR
Bits	Name	Description
31:8		Reserved for future use.
7:0	FNLSB	Frame number LSB. Least significant eight bits of frame number associated with last received SOF. Reset to all zeros.

8.5.24 UDC Byte Count Register 2

(UBCR2)

The Byte-Count Register maintains the remaining byte count in the active buffer of OUT endpoint2.

8.5.24.1 Endpoint 2 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read. The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:		UBCR2																								
Hex Offset Address:		0 x C800B068								Reset Hex Value:		0x00000000														
Register Description:		Universal Serial Bus Device Endpoint 2 Byte Count																								
Access: Read-Only																										
Bits																										
31																	8	7								0
(Reserved)																		BC[7:0]								
X																		0	0	0	0	0	0	0	0	
Resets (Above)																										



Register		UBCR2
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1)

8.5.25 UDC Byte Count Register 4

(UBCR4)

The Byte-Count Register maintains the remaining byte count in the active buffer of out Endpoint 4.

8.5.25.1 Endpoint 4 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count + 1.

Register Name:		UBCR4																									
Hex Offset Address:		0 x C800B06C								Reset Hex Value:		0x00000000															
Register Description:		Universal Serial Bus Device Endpoint 4 Byte Count																									
Access: Read-Only																											
Bits																											
31																		8	7								0
(Reserved)																BC[7:0]											
X																0	0	0	0	0	0	0	0	0			
Resets (Above)																											

Register		UBCR4
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

8.5.26 UDC Byte Count Register 7

(UBCR7)

The Byte-Count Register maintains the remaining byte count in the active buffer of out endpoint 7.



8.5.26.1 Endpoint 7 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count + 1.

Register Name:		UBCR7																							
Hex Offset Address:		0 x C800B070								Reset Hex Value:		0x00000000													
Register Description:		Universal Serial Bus Device Endpoint 7 Byte Count																							
Access: Read-Only																									
Bits																									
31																8	7								0
(Reserved)														BC[7:0]											
X														0	0	0	0	0	0	0	0				
Resets (Above)																									

Register		UBCR7
Bits	Name	Description
31:0		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

8.5.27 UDC Byte Count Register 9

(UBCR9)

The Byte-Count Register maintains the remaining byte count in the active buffer of out Endpoint 9.

8.5.27.1 Endpoint 9 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count + 1.



Register Name:		UBCR9																																		
Hex Offset Address:		0 x C800B074								Reset Hex Value:		0x00000000																								
Register Description:		Universal Serial Bus Device Endpoint 9 Byte Count																																		
Access: Read-Only																																				
Bits																																				
31																										8	7									0
(Reserved)																				BC[7:0]																
X																				0	0	0	0	0	0	0	0	0								
Resets (Above)																																				

Register		UBCR9
Bits	Name	Description
31:0		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

8.5.28 UDC Byte Count Register 12

(UBCR12)

The byte count register maintains the remaining byte count in the active buffer of out endpoint 12.

8.5.28.1 Endpoint 12 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count + 1.

Register Name:		UBCR12																																				
Hex Offset Address:		0 x C800B078												Reset Hex Value:		0x00000000																						
Register Description:		Universal Serial Bus Device Endpoint 12 Byte Count																																				
Access: Read-Only																																						
Bits																																						
31																										8	7											0
(Reserved)																				BC[7:0]																		
X																				0	0	0	0	0	0	0	0	0										
Resets (Above)																																						



Register		UBCR12
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

8.5.29 UDC Byte Count Register 14

(UBCR14)

The Byte-Count Register maintains the remaining byte count in the active buffer of out Endpoint 14.

8.5.29.1 Endpoint 14 Byte Count (BC[7:0])

The byte count is updated after each byte is read. When software receives an interrupt that indicates the endpoint has data, it can read the byte count register to determine the number of bytes that remain to be read.

The number of bytes that remain in the input buffer is equal to the byte count +1.

Register Name:		UBCR14																										
Hex Offset Address:		0 x C800B07C								Reset Hex Value:		0x00000000																
Register Description:		Universal Serial Bus Device Endpoint 14 Byte Count																										
Access: Read-Only																												
Bits																												
31																		8	7									0
(Reserved)																BC[7:0]												
X																0	0	0	0	0	0	0	0	0				
Resets (Above)																												

Register		UBCR14
Bits	Name	Description
31:8		(Reserved)
7:0	BC	Byte Count (read-only). Number of bytes in the FIFO is Byte Count plus 1 (BC+1).

8.5.30 UDC Endpoint 0 Data Register

(UDDR0)

The UDC Endpoint 0 Data Register is an 16-entry by 8-bit bidirectional FIFO. When the host transmits data to the UDC Endpoint 0, the Intel XScale processor reads the UDC Endpoint 0 Register to access the data.

When the UDC sends data to the host, the Intel XScale processor writes the data to be sent in the UDC Endpoint 0 Register. The Intel XScale processor can only read and write the FIFO at specific points in a control sequence.



The direction that the FIFO flows is controlled by the UDC. Normally, the UDC is in an idle state, waiting for the host to send commands. When the host sends a command, the UDC fills the FIFO with the command from the host and the Intel XScale processor reads the command from the FIFO when it arrives. The only time the Intel XScale processor may write the endpoint 0 FIFO is after a valid command from the host is received and it requires a transmission in response.

Register Name:		UDDR0																																					
Hex Offset Address:		0 x C800B080																Reset Hex Value:		0x00000000																			
Register Description:		Universal Serial Bus Device Endpoint 0 Data Register																																					
Access: Read/Write																																							
Bits																																							
31																										8												0	
(Reserved)																								(Data)															
X																																							
Resets (Above)																																							

Register		UDDR0			
Bits	Name	Description	Read Access	Write Access	
31:8		Reserved for future use.			
7:0	DATA	Top/bottom of endpoint 0 FIFO data. Read Bottom of endpoint 0 FIFO data. Write Top of endpoint 0 FIFO data.	Bottom of Endpoint 0 FIFO	Top of Endpoint 0 FIFO	

8.5.31 UDC Data Register 1

(UDDR1)

Endpoint 1 is a double-buffered bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale processor writes. Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:		UDDR1																								
Hex Offset Address:		0 x C800B100										Reset Hex Value:		0x00000000												
Register Description:		Universal Serial Bus Device Endpoint 1 Data Register																								
Access: Write																										
Bits																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																		(8-Bit Data)								
X																		0	0	0	0	0	0	0	0	0
Resets (Above)																										



Register		UDDR1
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.32 UDC Data Register 2

(UDDR2)

Endpoint 2 is a double-buffered bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to endpoint 2.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 2.

Register Name:		UDDR2																								
Hex Offset Address:		0 x C800B180								Reset Hex Value:		0x00000000														
Register Description:		Universal Serial Bus Device Endpoint 2 Data Register																								
Access: Read																										
Bits																										
31																	8	7								0
(Reserved)																(8-Bit Data)										
X																0	0	0	0	0	0	0	0	0		
Resets (Above)																										

Register		UDDR2
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

8.5.33 UDC Data Register 3

(UDDR3)

Endpoint 3 is a double-buffered isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.



Register Name:		UDDR3																															
Hex Offset Address:		0 x C800B200								Reset Hex Value:		0x00000000																					
Register Description:		Universal Serial Bus Device Endpoint 3 Data Register																															
Access: Write																																	
Bits																																	
31														16	15									8	7								0
(Reserved)														(8-Bit Data)																			
X														0	0	0	0	0	0	0	0	0											
Resets (Above)																																	

Register		UDDR3
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.34 UDC Data Register 4

(UDDR4)

Endpoint 4 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 4.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 4.

Register Name:		UDDR4																							
Hex Offset Address:		0 x C800B400								Reset Hex Value:		0x00000000													
Register Description:		Universal Serial Bus Device Endpoint 4 Data Register																							
Access: Read																									
Bits																									
31																8	7								0
(Reserved)														(8-Bit Data)											
X														0	0	0	0	0	0	0	0				
Resets (Above)																									

Register		UDDR4
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.



8.5.35 UDC Data Register 5

(UDDR5)

Endpoint 5 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 5 to determine interrupt conditions. The UDC cannot initiate the transaction.

Register Name:		UDDR5																									
Hex Offset Address:		0 x C800B008								Reset Hex Value:		0x00000000															
Register Description:		Universal Serial Bus Device Endpoint 5 Data Register																									
Access: Write.																											
Bits																											
31																		8	7								0
(Reserved)																		(8-Bit Data)									
X																		0	0	0	0	0	0	0	0		
Resets (Above)																											

Register		UDDR5
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.36 UDC Data Register 6

(UDDR6)

Endpoint 6 is a double-buffered, bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:		UDDR6																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Hex Offset Address:		0 x C800B600												Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
Register Description:		Universal Serial Bus Device Endpoint 6 Data Register																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Access: Write																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
Bits																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																										8	7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										



Register		UDDR6
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.37 UDC Data Register 7

(UDDR7)

Endpoint 7 is a double-buffered, bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt request as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to endpoint 7.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 7.

Register Name:		UDDR7																											
Hex Offset Address:		0 x C800B680								Reset Hex Value:		0x00000000																	
Register Description:		Universal Serial Bus Device Endpoint 7 Data Register																											
Access: Read																													
Bits																													
31																	8	7								0			
(Reserved)																(8-Bit Data)													
X																0	0	0	0	0	0	0	0	0	0	0			
Resets (Above)																													

Register		UDDR7
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.

8.5.38 UDC Data Register 8

(UDDR8)

Endpoint 8 is a double-buffered, isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale processor writes.

Because it is double buffered, up to two packets of data may be loaded for transmission.



Register Name:		UDDR8																									
Hex Offset Address:		0 x C800B700								Reset Hex Value:		0x00000000															
Register Description:		Universal Serial Bus Device Endpoint 8 Data Register																									
Access: Write																											
Bits																											
31																		8	7								0
(Reserved)																		(8-Bit Data)									
X																		0	0	0	0	0	0	0	0		
Resets (Above)																											

Register		UDDR8
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.39 UDC Data Register 9

(UDDR9)

Endpoint 9 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt request when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 9.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 9.

Register Name:		UDDR9																									
Hex Offset Address:		0 x C800B900												Reset Hex Value:		0x00000000											
Register Description:		Universal Serial Bus Device Endpoint 9 Data Register																									
Access: Read																											
Bits																											
31																		8	7								0
(Reserved)																			(8-Bit Data)								
X																			0	0	0	0	0	0	0	0	
Resets (Above)																											

Register		UDDR9
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being read.



8.5.40 UDC Data Register 10

(UDDR10)

Endpoint 10 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 10 to determine interrupt conditions. The UDC cannot initiate the transaction.

Register Name:		UDDR10																																		
Hex Offset Address:		0 x C800B0C0								Reset Hex Value:		0x00000000																								
Register Description:		Universal Serial Bus Device Endpoint 10 Data Register																																		
Access: Write																																				
Bits																																				
31																										8	7									0
(Reserved)																		(8-Bit Data)																		
X																		0	0	0	0	0	0	0	0	0										
Resets (Above)																																				

Register		UDDR10
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.41 UDC Data Register 11

(UDDR11)

Endpoint 11 is a double-buffered, bulk IN endpoint that is 64 bytes deep. Data can be loaded via direct Intel XScale processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.

Register Name:	UDDR11																									
Hex Offset Address:	0 x C800BB00												Reset Hex Value:	0x00000000												
Register Description:	Universal Serial Bus Device Endpoint 11 Data Register																									
Access: Write																										
Bits																										
31																		8	7							0
(Reserved)																		(8-Bit Data)								
X																		0	0	0	0	0	0	0	0	
Resets (Above)																										



Register		UDDR11
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.42 UDC Data Register 12

(UDDR12)

Endpoint 12 is a double-buffered, bulk OUT endpoint that is 64 bytes deep. The UDC will generate an interrupt request as soon as the EOP is received.

Since it is double-buffered, up to two packets of data may be ready. Via direct read from the Intel XScale processor, the data can be removed from the UDC. If one packet is being removed and the packet behind it has already been received, the UDC will issue a NAK to the host the next time it sends an OUT packet to Endpoint 12.

This NAK condition will remain in place until a full packet space is available in the UDC at Endpoint 12.

Register Name:		UDDR12																								
Hex Offset Address:		0 x C800BB80								Reset Hex Value:		0x00000000														
Register Description:		Universal Serial Bus Device Endpoint 12 Data Register																								
Access: Read																										
Bits																										
31																	8	7								0
(Reserved)																(8-Bit Data)										
X																0	0	0	0	0	0	0	0	0		
Resets (Above)																										

Register		UDDR11
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.43 UDC Data Register 13

(UDDR13)

Endpoint 13 is a double-buffered, isochronous IN endpoint that is 256 bytes deep. Data can be loaded via direct Intel XScale processor writes.

Because it is double-buffered, up to two packets of data may be loaded for transmission.



Register Name:		UDDR13																																		
Hex Offset Address:		0 x C800BC00								Reset Hex Value:		0x00000000																								
Register Description:		Universal Serial Bus Device Endpoint 13 Data Register																																		
Access: Write																																				
Bits																																				
31																										8	7									0
(Reserved)																		(8-Bit Data)																		
X																		0	0	0	0	0	0	0	0	0										
Resets (Above)																																				

Register		UDDR13
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.

8.5.44 UDC Data Register 14

(UDDR14)

Endpoint 14 is a double-buffered, isochronous OUT endpoint that is 256 bytes deep. The UDC generates an interrupt request when the EOP is received.

Because it is double-buffered, up to two packets of data may be ready. The data can be removed from the UDC via a direct read from the Intel XScale processor. If one packet is being removed and the packet behind it has already been received, the UDC issues a NAK to the host the next time it sends an OUT packet to Endpoint 14.

This NAK condition remains in place until a full packet space is available in the UDC at Endpoint 14.

Register Name:	UDDR14																									
Hex Offset Address:	0 x C800BE00												Reset Hex Value:	0x00000000												
Register Description:	Universal Serial Bus Device Endpoint 14 Data Register																									
Access: Read																										
Bits																										
31																		8	7							0
(Reserved)																			(8-Bit Data)							
X																			0	0	0	0	0	0	0	0
Resets (Above)																										

Register		UDDR14
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.



8.5.45 UDC Data Register 15

(UDDR15)

Endpoint 15 is an interrupt IN endpoint that is 8 bytes deep. Data must be loaded via direct Intel XScale processor writes.

Because the USB system is a host-initiator model, the host must poll Endpoint 15 to determine interrupt conditions. The UDC cannot initiate the transaction.

Register Name:		UDDR15																									
Hex Offset Address:		0 x C800B0E0								Reset Hex Value:		0x00000000															
Register Description:		Universal Serial Bus Device Endpoint 15 Data Register																									
Access: Write																											
Bits																											
31																		8	7								0
(Reserved)																		(8-Bit Data)									
X																		0	0	0	0	0	0	0	0		
Resets (Above)																											

Register		UDDR15
Bits	Name	Description
31:8		Reserved for future use.
7:0	DATA	Top of endpoint data currently being loaded.



9.0 USB 2.0 Host Controller

9.1 Overview

This unit describes the USB Host controller functionality for the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The functions being performed are defined by the USB 2.0 specification, further information can be found at www.usb.org.

Not all features defined by the USB 2.0 specification are supported by the IXP45X/IXP46X network processors. The following is a partial list of supported features:

- EHCI register interface
- Host function
- Low-speed interface
- Full-speed interface

The following is a partial list of USB 2.0 features *not* supported by the IXP45X/IXP46X network processors:

- High-speed interface
- Device function
- OTG function

The USB Host core is a standards-based “Serial Interface Engine” for implementing a USB interface in compliance with the USB 2.0 specification. The register and data structure interfaces to the core are based on the Enhanced Host Controller Interface (EHCI) specification from the Intel Corporation. By leveraging industry standards throughout the core, the compatibility and quality of the final product are insured.

The USB Host core is designed to make efficient use of the system resources in this design. The 32-bit system bus interface contains a chaining DMA (Direct Memory Access) engine. This DMA engine reduces the interrupt load on the application processor and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface.

Both this USB-host interface and the USB-device interface can be used at the same time. (For more information on the USB-device interface, see [Chapter 8.0, “USB 1.1 Device Controller”](#) on [page 278](#).)

9.2 USB

The Universal Serial Bus (USB) is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation.



USB software provides a uniform view of the system for all application software, hiding implementation details making application software more portable. It manages the dynamic attach and detach of peripherals.

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller. The Host Controller may be implemented in a combination of hardware, firmware, or software.

There may be multiple USB devices in any system such as hubs, joysticks, speakers, printers, etc. USB devices present a standard USB interface in terms of comprehension, response, and standard capability.

The host initiates transactions to specific peripherals, while the device responds to control transactions. The device sends and receives data to and from the host using a standard USB data format. The early versions of the USB specification (USB 1.0 and 1.1) operated at 12 Mbps or 1.5 Mbps.

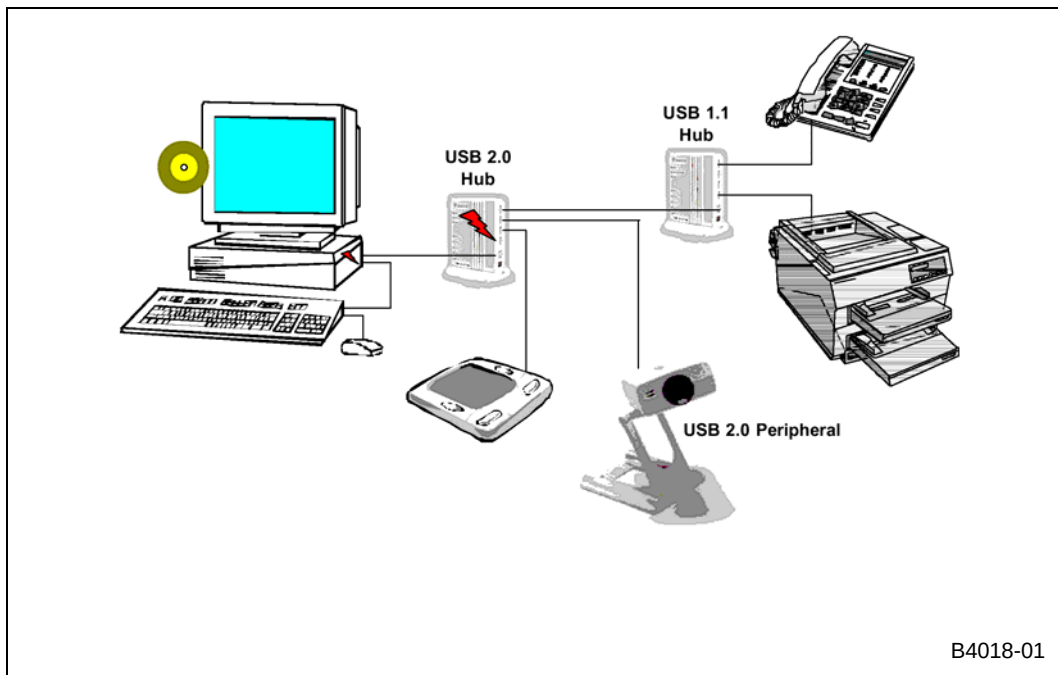
9.3 USB 2.0

The USB 2.0 specification supersedes the earlier versions of the USB specification. USB 2.0 offers a high-speed protocol which offers the user a larger bandwidth and increases data throughput by a factor of 40. All the peripherals used with the previous versions of USB work perfectly with USB 2.0 while also offering a larger choice of higher performance peripherals, such as video-conferencing cameras, next-generation scanners and printers, a fast storage device, etc.

In addition to the 1.5-Mbps and 12-Mbps data rates of USB 1.1, the evolution from USB 1.1 to USB 2.0 adds an additional data rate of 480 Mbps. Existing USB 1.1 connectors and full-speed cables support the higher speeds of the USB 2.0 without any changes. USB 2.0 specifies a microframe, which is 1/8th of a 1-ms frame allowing small buffers even at the high data rate.

USB 2.0 system configurations look identical to the end user to configurations based on earlier revisions of the USB specification. However, the end user will need to distinguish between USB 2.0 hubs and USB 1.1 hubs to optimize the placement of USB 2.0 high-speed devices. A USB 2.0 hub accepts high-speed transactions at a faster rate but also matches the data rate of the peripherals, while the USB 1.1 hub will only support the lower 12-Mbps and 1.5-Mbps data rates.

Figure 36. Example USB 2.0 System Configuration



The roles of the components of the USB 2.0 system have minor changes from the roles in a USB 1.1 system. However, current peripheral products work with the USB 2.0 system without any changes. Additional performance is not required for many of the human interface devices such as mice, keyboards, and game pads. Both USB 1.1 and USB 2.0 devices will operate in a USB 2.0 system while opening up the possibilities of exciting new higher bandwidth peripherals.

For additional information, refer to the USB 2.0 specification.

Note: The USB Host functionality for the IXP45X/IXP46X network processors is USB 2.0 specification-compliant but does not support the 480-Mbps, high-speed protocol.

9.4 Feature List

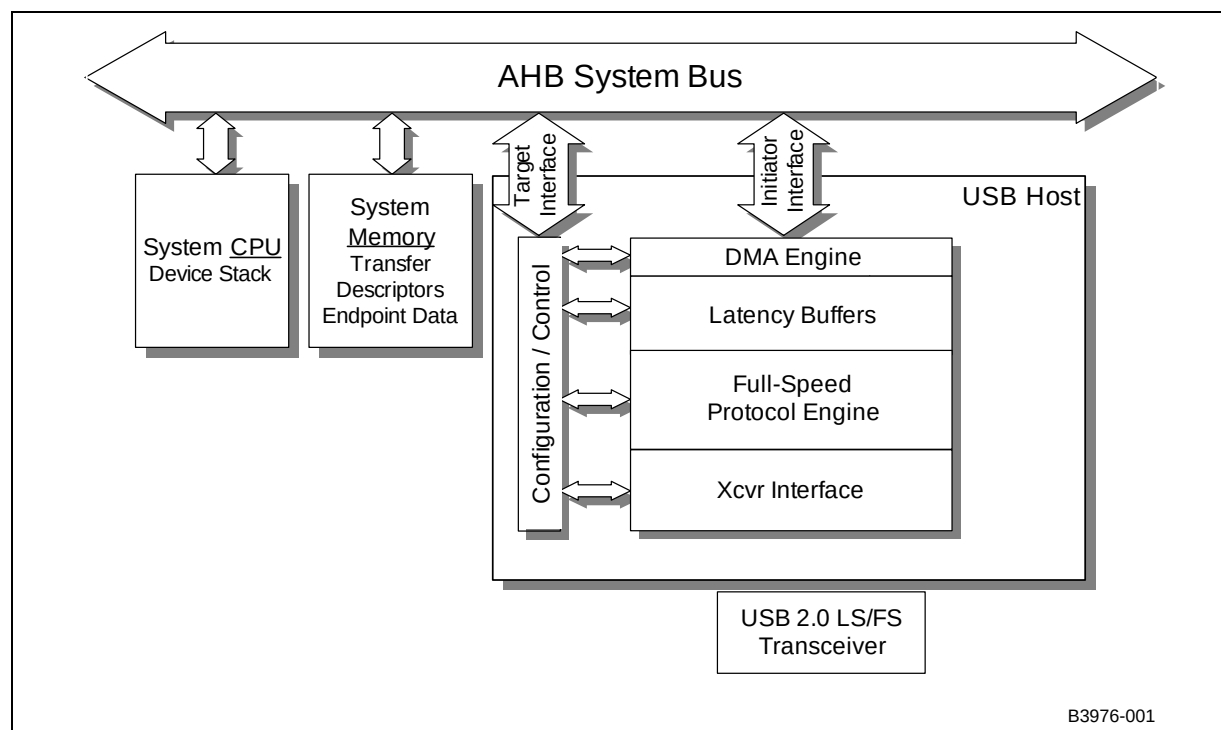
- Intel EHCI host controller. The USB host controller registers and data structures are compliant to Intel® EHCI specification. Device controller registers and data structures are implemented as extensions to the EHCI programmers interface. Further information can be found on Intel's web site at: <http://www.intel.com/technology/usb/ehcispec.htm>
- Directly connected USB legacy (USB 1.1) Full and Low speed devices without a companion USB 1.1 host controller or host controller driver software using EHCI standard data structures.
- Configurable dual port RAM buffers isolate memory latency on the system bus from the timing requirements of the USB.



9.5 Block Diagram

Figure 37 shows the top-level block diagram for the module. In this case, the system CPU is the Intel XScale® Processor and the system memory is whatever system memory the USB block has its DMA engine pointed to. In this case, note that the USB 2.0 compliance is restricted to a legacy full-speed protocol and that the system memory is restricted to Memory Controller Unit accesses only.

Figure 37. Top-Level Block Diagram



9.6 Theory of Operation

9.6.1 Software Model

The Host Stack provides a layered software architecture to control all aspects of a USB bus system. The Host Controller Device (HCD) interface controls the functions of an embedded EHCI host controller. The USB driver layer provides all the USB driver functions to enumerate, manage and schedule a USB bus system, while the upper layers of the stack support standard USB device class interfaces to the device drives running on your embedded system. Detailed information can be obtained by referring to the USB 2.0 specification located at www.usb.org.

9.6.2 Host Data Structure

The host data structures are used to communicate control, status, and data between software and the Host Controller. The **Periodic Frame List** is an array of pointers for the periodic schedule. A sliding window on the Periodic Frame List is used. The **Asynchronous Transfer List** is where all the control and bulk transfers are managed. The Host API incorporates and abstracts for the application developer all of the information contained in the host operational model.



Figure 38. Periodic Schedule Organization

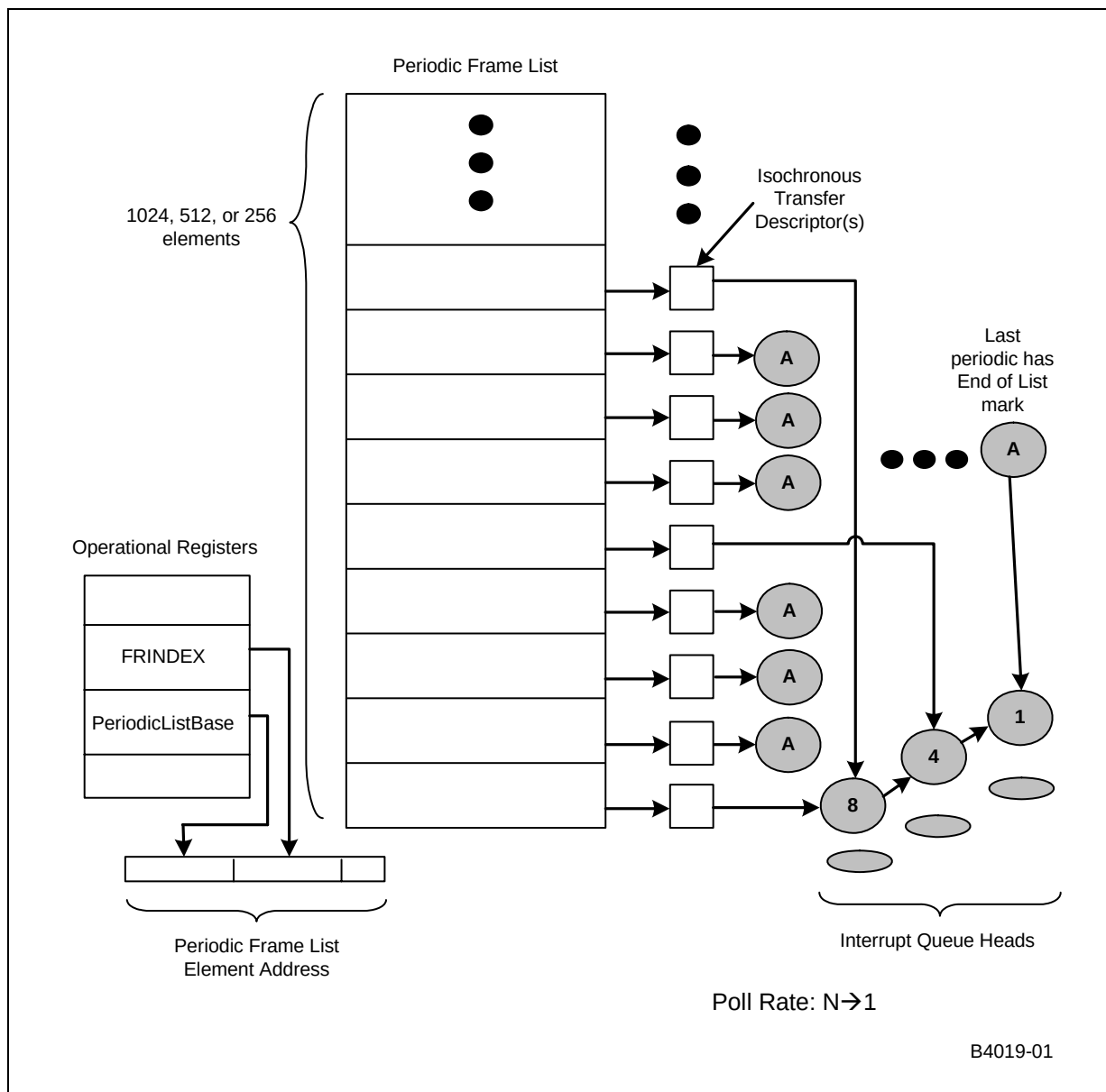
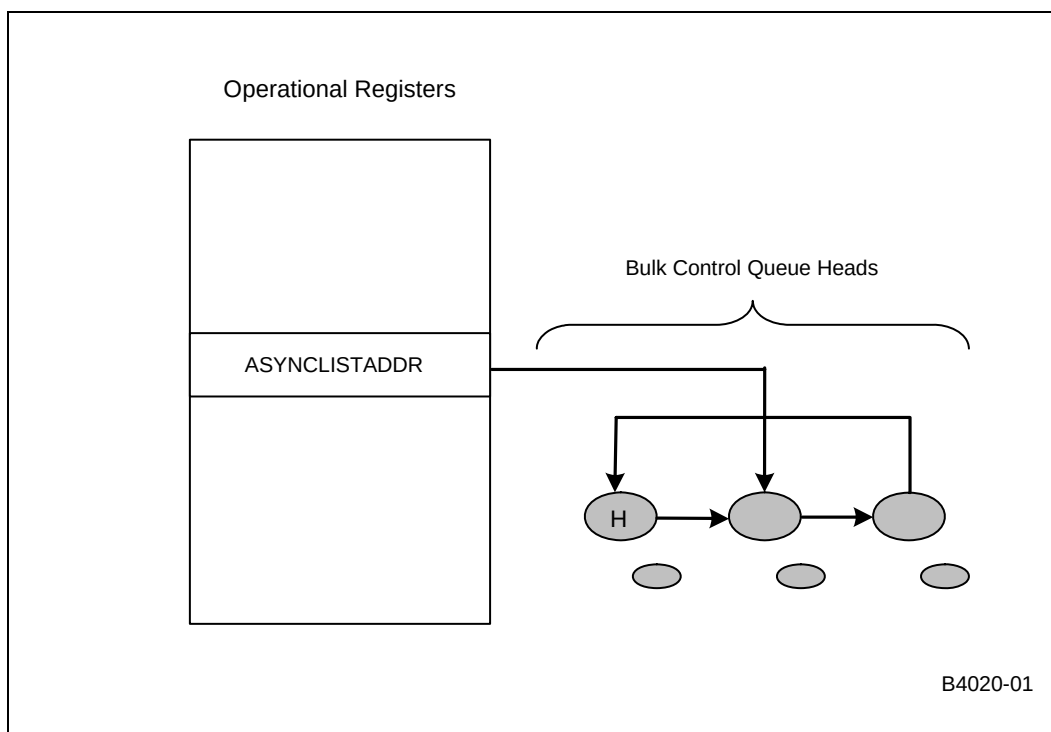




Figure 39. Asynchronous Schedule Organization

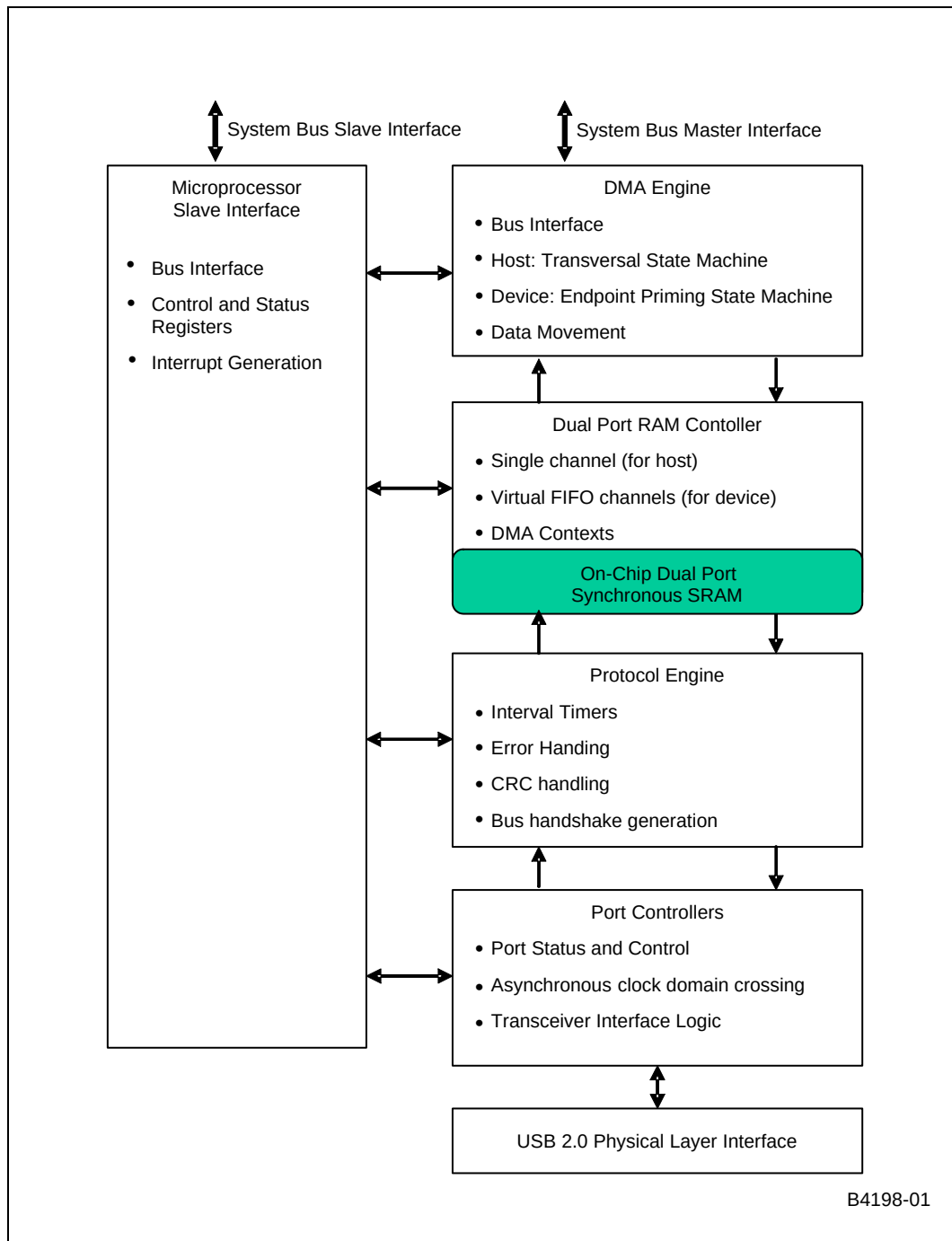




9.6.3 Hardware Model

9.6.3.1 Block Diagram

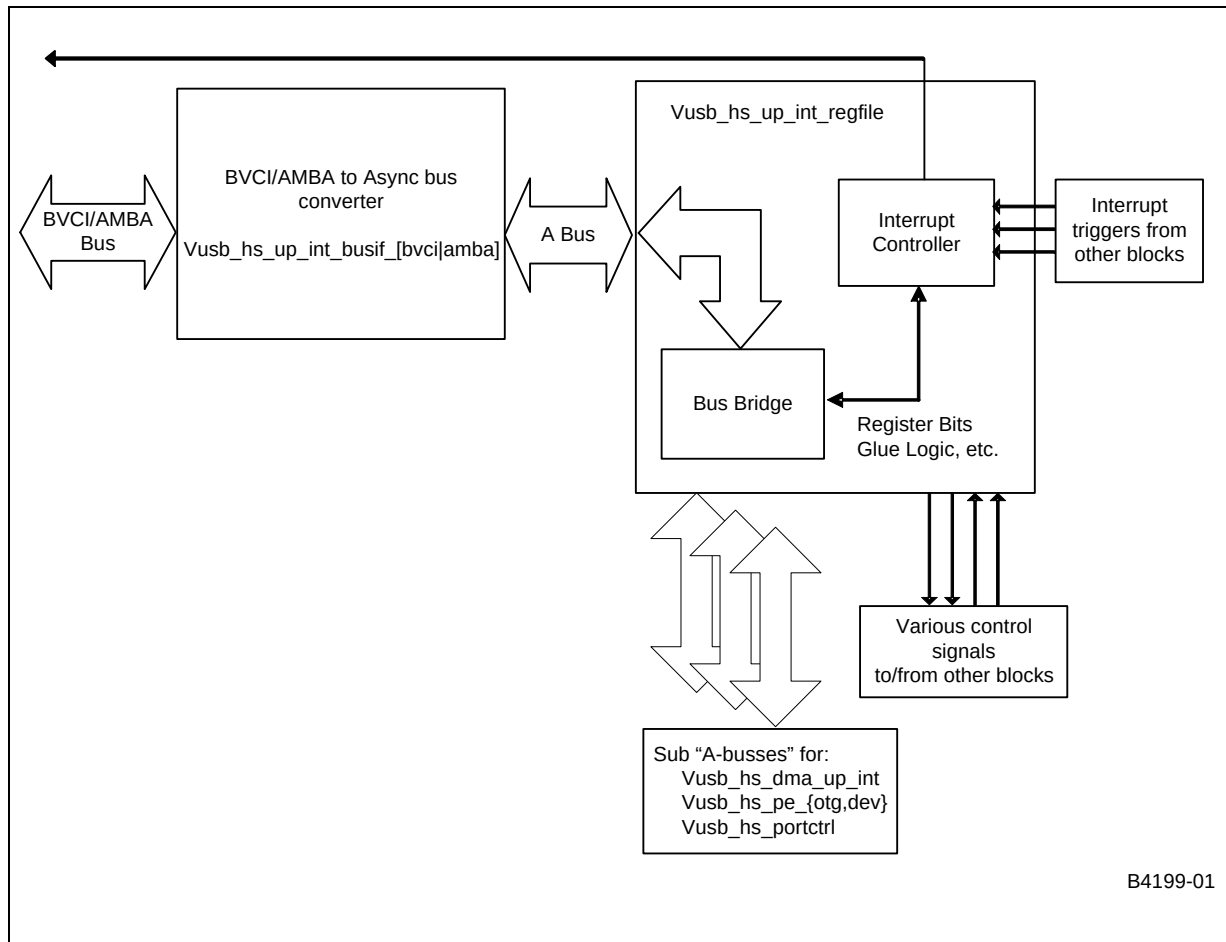
Figure 40. Block Diagram





9.6.3.2 Microprocessor Interface

Figure 41. Microprocessor Interface Block Diagram



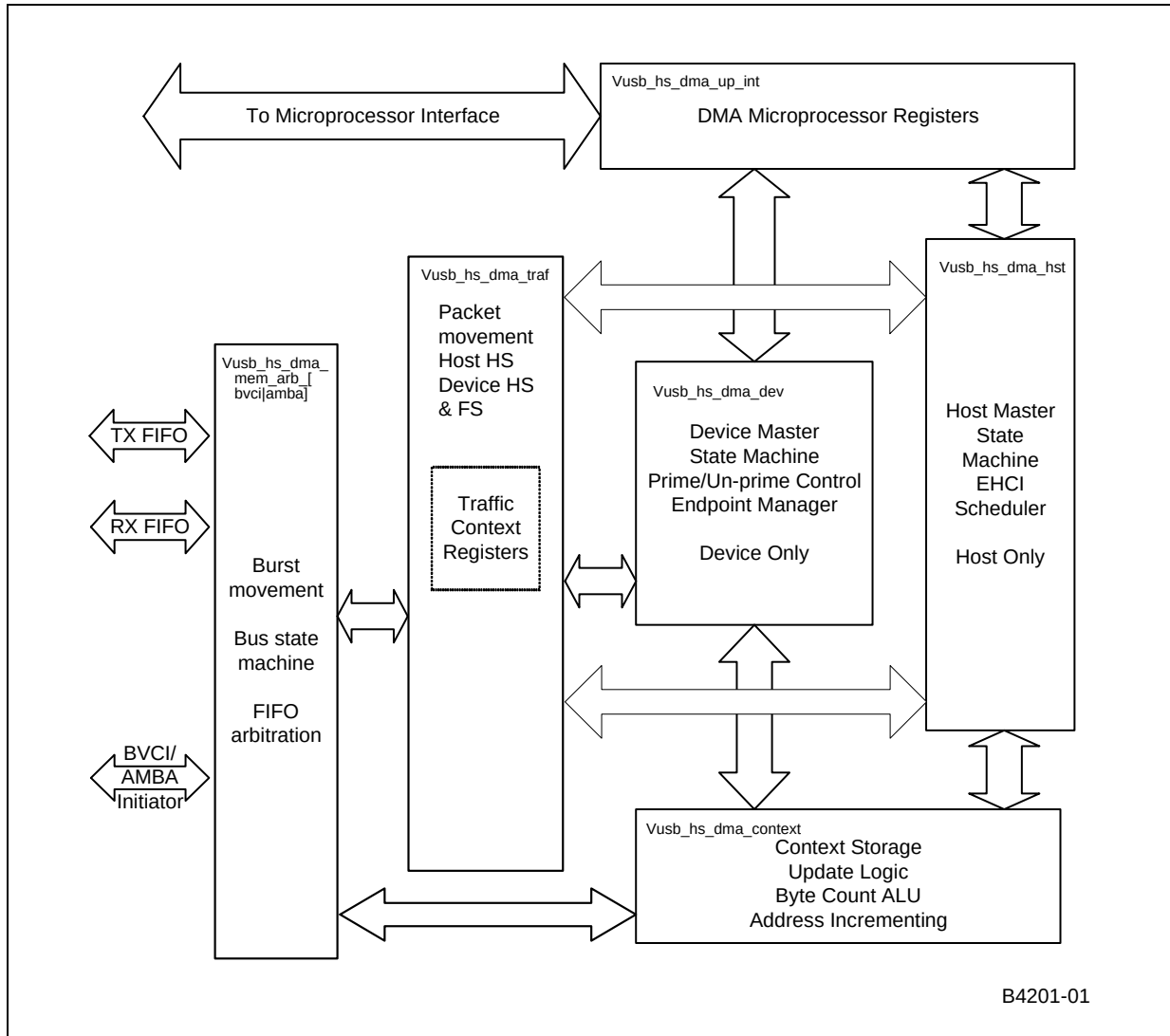
The microprocessor interface block provides an AMBA target (slave) interface on the AHB. A simple synchronous bus interface signaling is used for the microprocessor bus allowing the core to be easily connected to the system bus.

The Microprocessor interface block contains all the control and status registers that allow a processor to interface to the USB Host core. These registers allow a microprocessor to control the configuration of the core, ascertain the capabilities of the core and control the core in operation for host mode.

Two groups of registers exist in the interface. The USB host controller registers are compatible with the USB host controller registers defined in the Intel™ EHCI specification.

9.6.3.3 DMA Engine

Figure 42. DMA Engine Block Diagram



The **DMA Engine Block** presents a bus initiator (master) interface to the system memory bus. It is responsible for moving all of the data to be transferred over the USB between the UHS Host core and buffers in system memory. Like the microprocessor interface block the DMA block uses the AHB protocol.

The DMA controller must access both control information and packet data from system memory. The control information is contained in link list based queue structures. The DMA controller has state machines that are able to parse all of the data structures defined in this controller specification. In host mode, the data structures are from the EHCI specification and represent queues of transfers to be performed by the host controller, including the split transaction requests that allow an EHCI controller to direct packets to Low and Full speed devices.

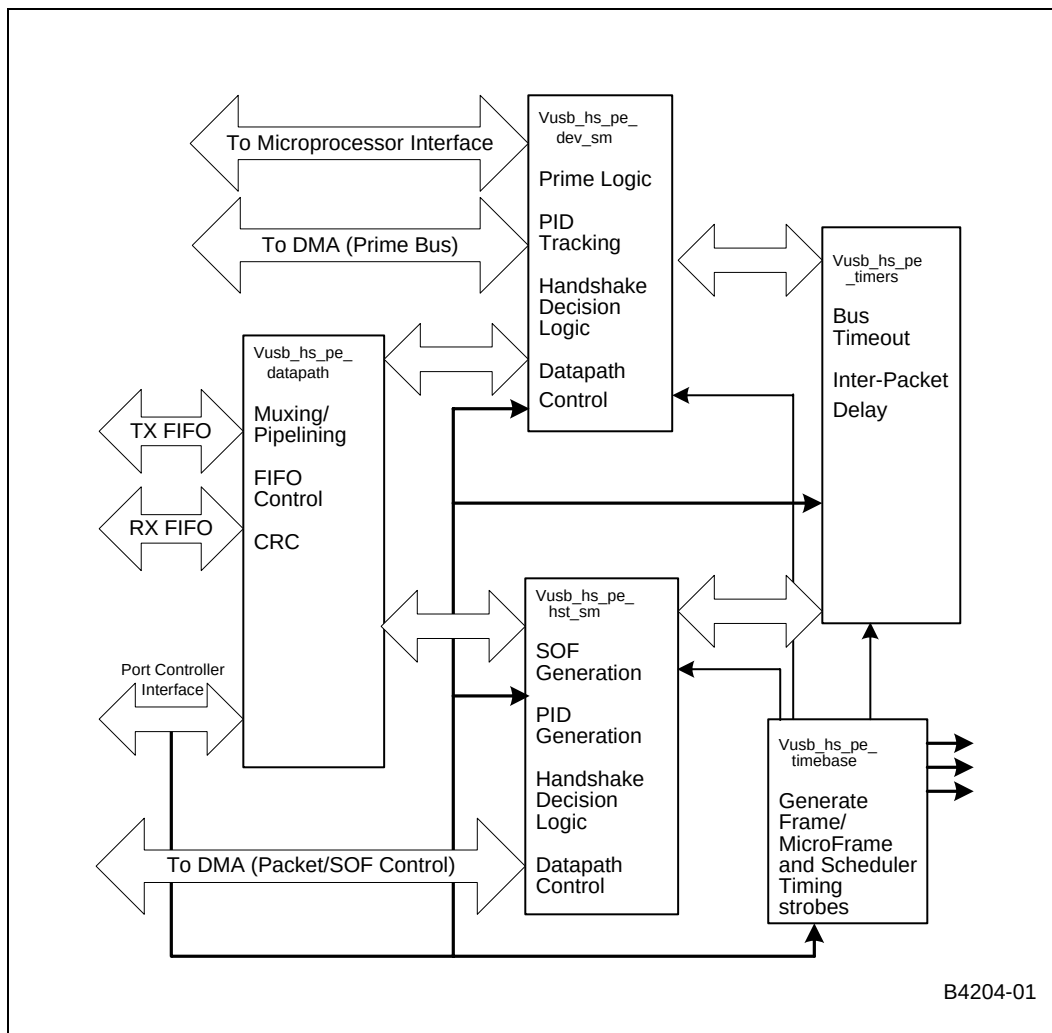


9.6.3.4 Dual Port RAM Controller

The **Dual Port RAM Controller** is used for context information and to build configurable FIFOs between the Protocol Engine block and the DMA controller. These FIFOs decouple the system processor memory bus request from the extremely tight timing required by the USB itself.

9.6.3.5 Protocol Engine

Figure 43. Protocol Engine Block Diagram



The **Protocol Engine** parses all the USB tokens and generates the response packets. It is responsible for all error checking, check field generation, formatting all the handshake, ping and data response packets on the bus, and for any signals that must be generated based on a USB based time frame. In host mode, the Protocol engine also generates all of the token packets required by the USB protocol. The Protocol engine contains several sub-functions:

- The token state machines track all of the tokens on the bus and filter the traffic based on the address and endpoint information in the token. In host mode, these state machines also generate the tokens required for data transfer and bus control.

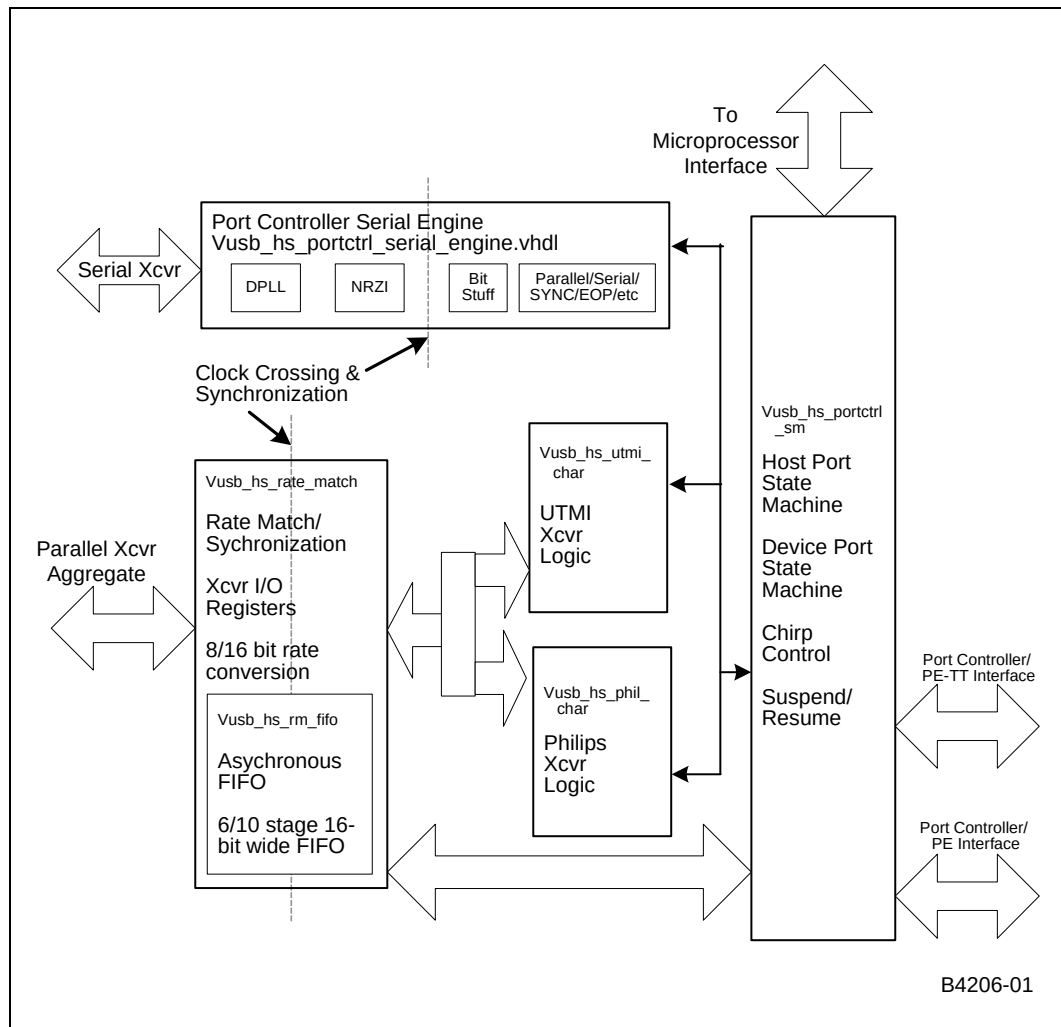


- The **CRC5** and **CRC16** CRC generator/checker circuits check and generate the CRC check fields for the token and data packets.
- The data and handshake state machines generate any responses required on the USB and move the packet data through the dual port memory FIFOs to the DMA controller block.
- The Interval timers provide timing strobes that identify important bus timing events: the bus time-out interval, the microframe interval, the start of frame interval, and the bus reset, resume, and suspend intervals.
- Reports all transfer status to the DMA engine.

9.6.3.6 Port Controller

The **Port Controller** block interfaces to the full-speed transceiver or any UTMI compatible transceiver macro-cell core. The primary function of the **Port Controller** block is to isolate the rest of the core from the transceiver and to move all of the transceiver signaling into the primary clock domain of the USB core. This allows the USB core to run synchronously with the system processor and its associated resources.

Figure 44. Port Controller Block Diagram





9.6.3.7 System Bus Interface

The USB Host core interfaces to the system memory and processing resources using an AMBA AHB. This provides the features required for high performance/high clock frequency systems including:

- Synchronous Single clock edge operation
- Non-tristate on-chip implementation
- Initiator (master) and Target (slave) interfaces
- Burst transfers

9.7 System Level Issues and Core Configuration

The core is designed to be part of a larger system containing a processor, a bus architecture, and a memory system. This section is intended to provide information about the core that is required by a system designer to construct a complete system.

9.7.1 Configuration Constants

The following table summarizes the configuration controls.

Table 121. Configuration Controls (Sheet 1 of 2)

Constant	Description	Supported Values
VUSB_HS_RESET_TYPE	Controls the reset type.	0=Synchronous, 1=Asynchronous.
VUSB_HS_RESET_OPTIONAL	Adds resets to register files and other groups of registers where a reset is not functionally required. This is only used in the MPH design and has no effect on DEV, SPH, or OTG. The MPH product has this option because a large number of flops are affected and adding resets to those flops has considerable increase on the gate count. All registers have reset in DEV, SPH, and OTG.	0=No extra resets 1=Reset all flops
VUSB_HS_CLOCK_CONFIGURATION	Controls the clocking architecture.	[0,1,2,3] See following section on System Clocking.
VUSB_HS_NUM_PORT	Configures the number of ports for the multi-port host product. All other products must leave this value at 1.	[1-8] Number of downstream host ports.
VUSB_HS_PHY16_8	Controls the data interface to the UTMI transceiver. Should be set to match the transceiver connected to the core.	0= 8-bit data bus [60MHz] 1=16-bit data bus [30MHz] 2=Software Controlled reset to 8-bit data bus [60MHz] 3=Software Controlled reset to 16-bit data bus [30MHz]
VUSB_HS_PHY_MODE	Controls which USB transceiver interface is being used.	0 = UTMI Only 1 = Philips (w/Serial FS) 2 = Serial FS Only 3 = Software controlled reset to UTMI 4 = Software controlled reset to Philips 5 = Software controlled reset to Serial FS
VUSB_HS_SERIAL_MODE	Controls the usage of the xcvr_ser_se0_vmo output. This constant is only used when the serial engine is used	0 = xcvr_ser_se0_vmo is se0 1 = xcvr_ser_se0_vmo is vmo

**Table 121. Configuration Controls (Sheet 2 of 2)**

Constant	Description	Supported Values
VUSB_HS_BANDWIDTH_TESTING	Controls the testing of the core. This constant should always be zero.	0=Normal Bandwidth Testing 1=Bandwidth Starved Testing (subset of Normal Bandwidth Testing) Used for system clock less than 30 MHz.
VUSB_HS_DEV_EP	Controls the maximum number of endpoints supported by the core.	Integer values between 2 and 16. Set this to 1 for single and multi-port host-only products.
VUSB_HS_NUM_PORT	Controls the number of downstream ports in a host implementation.	Integer values between 1 and 8. Set this to 1 for non-multi-port products.
VUSB_HS_TT_PERIODIC_CONTEXTS	USB 2.0 specification requires a hub Transaction Translator to have 16 periodic contexts. However, for some host applications 4 may be adequate and a gate count savings can be realized.	4 or 16.
VUSB_HS_RX_DEPTH	Controls the size of the receive latency buffer.	Powers of 2 from 8 to 2048.
VUSB_HS_RX_BURST	Controls the Bus burst size for Rx DMA data transfers.	Integer values from 4 to 128.
VUSB_HS_TX_CHAN	Controls the size of each of the transmit latency buffers.	Powers of 2 from 16 to 128.
VUSB_HS_TX_BURST	Controls the Bus burst size for Tx DMA data transfers.	Integer values from 4 to 128.
VUSB_HS_TX_LOCAL_CONTEXT_REGISTERS	Determines if device transmit context registers are implemented as a register file or stored in the TX-FIFO.	1=Implement device transmit contexts in registers. (Removes need for read port A on TX-FIFO) 0=Store device transmit contexts in TX FIFO. (Smaller total gate count but uses 4 words of VUSB_HS_TX_CHAN per endpoint)
VUSB_HS_HCVERSION	Software readable host silicon version.	Unsigned 16-bit integer.
VUSB_HS_DCVERSION	Software readable device silicon version.	Unsigned 16-bit integer.

9.8 Detailed Register Descriptions

The MMR registers for the USB Host are (for the most part) EHCI specification compliant, with exceptions as noted. The USB Host supports byte access to the MMR space and the default access is “little-endian” after power-up or hardware reset. The endian support for the MMR space is programmable which enables “big-endian” access. This is important for byte access support and it is expected that one of the very first operations being performed on the USB host is to correctly set the required endian mode. Please see the USBMODE register description for details.

Table 122. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only

**Table 122. Register Legend**

Attribute	Legend	Attribute	Legend
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Slave accesses from the controlling processor enables access to the configuration, control, and status registers. One function of the system address map is the registers base address, which must begin on a **DWord** (32-bit) boundary.

Configuration, control and status registers are divided into three categories:

- Identification registers — Identification Registers are used to declare the slave interface presence along with the complete set of the hardware configuration parameters. These are internal registers to the design and are not exposed to the user of the product.
- Capability registers — Static, read only capability registers define the software limits, restrictions, and capabilities of the host controller.
- Operational registers — Comprised of dynamic control or status registers that may be read only, read/write, or read/write to clear. The following sections define the use of these registers.

EHCI registers are listed alongside device registers to show the complementary nature of host and device control.

Please note that the USB core utilized within this system was designed to support both Host and Device operation. However, the Host functionality is the only item which was actually implemented and tested on this particular instantiation of the USB Core. Any attempt to utilize the USB Device functionality included within this core will result in unpredictable behavior.

Table 123. Interface Register Sets

Offset	Register Set	Explanation
000h to 0fch	Identification Registers	Identification registers are used to declare the slave interface presence and include a table of the hardware configuration parameters.
100h to 124h	Capability Registers	Capability registers specify the limits, restrictions, and capabilities of a host controller implementation. These values are used as parameters to the host driver.
140h to 1FCh	Operational Registers	Operational registers are used by the system software to control and monitor the operational state of the host controller.



9.9 Configuration, Control and Status Register Set

Table 124. Host Capability Registers (Sheet 1 of 2)

Offset	Size (Bytes)	Mnemonic	Register Name			Single Port Host (SPH)	
020h-0FCh	232	(Reserved)	N/A				
100h	1	CAPLENGTH	Capability Register Length			÷	
101h	1	(Reserved)	N/A				
102h	2	HCVERSION	Host Interface Version Number			÷	
104h	4	HCSPARAMS	Host Ctrl. Structural Parameters			÷	
108h	4	HCCPARAMS	Host Ctrl. Capability Parameters			÷	
10Ch – 11Fh	20	(Reserved)	N/A				
120h	2	(Reserved)					
122h	2	(Reserved)	N/A				
124h	4	DCCPARAMS	Device Ctrl. Capability Parameters				
128h – 13Ch	24	(Reserved)	N/A				
140h	4	USBCMD	USB Command			÷	
144h	4	USBSTS	USB Status			÷	
148h	4	USBINTR	USB Interrupt Enable			÷	
14Ch	4	FRINDEX	USB Frame Index			÷	
150h	4	(Reserved)	4G Segment Selector				
154h	4	PERIODICLISTBASE	Frame List Base Address			÷	
		Device Addr	USB Device Address				
158h	4	ASYNCLISTADDR	Next Asynchronous List Address			÷	
		Endpointlist Addr	Address at Endpoint list in memory				
160h	4	BURSTSIZE	Programmable Burst Size			÷	
164h	4	TXFILLTUNING	Host Transmit Pre-Buffer Packet Tuning			÷	
16Ch	4	N/A	(Reserved)				
170-17Ch	16	N/A	(Reserved)				

**Table 124. Host Capability Registers (Sheet 2 of 2)**

Offset	Size (Bytes)	Mnemonic	Register Name			Single Port Host (SPH)	
180h	4	CONFIGFLAG	Configured Flag Register			÷	
184h	4	PORTSC1	Port Status/Control 1			÷	
1A8h	4	USBMODE	USB Device Mode			÷	

9.10 Identification Registers

Identification registers are used to declare the slave interface presence and include a table of the hardware configuration parameters.

9.10.1 ID

Address: Base + 000h

Default Value: 0x0033_FA05

Attribute: Read Only

Size: 32 bits

The Identification register (ID) provides a simple way to determine if the USB 2.0 core is provided in the system. The ID register identifies the USB 2.0 core and its revision.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)								REVISION								1	1	NID						0	0	ID					

Table 125. Identification Register Fields

Field	Description
ID[5:0]	Configuration number. This number is set to 0x05 and indicates that the peripheral is the USB 2.0 core.
NID[5:0]	Ones complement version of ID[5:0].
REVISION[7:0]	Revision number of the core
(Reserved)	These bits are reserved and should be zero.

9.10.2 HWGENERAL

Address: Base + 004h

Default Value: 00000000_00000000_000000_1_010_00_0_01_1b

0x0000_0283

Attribute: Read Only

Size: 32 bits

General hardware parameters as defined in [Section 9.7, “System Level Issues and Core Configuration” on page 363](#).



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																						SM	PHYM			PHYW		BWT	CLKC		RT

Table 126. HWGENERAL – General Hardware Parameters: Fields

Field	Description
(Reserved)	These bits are reserved and should be zero.
SM	VUSB_HS_SERIAL_MODE
PHYM	VUSB_HS_PHY_MODE
PHYW	VUSB_HS_PHY16_8
BWT	VUSB_HS_BANDWIDTH_TESTING
CLKC	VUSB_HS_CLOCK_CONFIGURATION
RT	VUSB_HS_RESET_TYPE

9.10.3 HWHOST

Address: Base + 008h

Default Value: 10_02_00_01h

Attribute: Read Only

Size: 32 bits

Host hardware parameters as defined in [Section 9.7, “System Level Issues and Core Configuration” on page 363](#).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTPER								TTASY								(Reserved)								NPORT		HC					

Table 127. HWHOST – Host Hardware Parameters

Field	Description
TTPER	VUSB_HS_TT_PERIODIC_CONTEXTS
TTASY	VUSB_HS_TT_ASYNC_CONTEXTS
(Reserved)	These bits are reserved and should be zero.
NPORT	VUSB_HS_NUM_PORT-1
HC	VUSB_HS_HOST

9.10.4 HWDEVICE

Address: Base + 00Ch

Default Value: 0x0000_0009

Attribute: Read Only

Size: 32 bits

Device hardware parameters as defined in [Section 9.7, “System Level Issues and Core Configuration” on page 363](#).



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																										DEVEP			DC		

Table 128. HWDEVICE – Device Hardware Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
DEVEP	VUSB_HS_DEV_EP
DC	device capable; [VUSB_HS_DEV != 0]

9.10.5 HWTXBUF

Address: Base + 010h

Default Value: 80_04_06_04h

Attribute: Read Only

Size: 32 bits

TX buffer hardware parameters as defined in [Section 9.7, “System Level Issues and Core Configuration” on page 363](#).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXLCR	(Reserved)							TXCHANADD							TXADD							TCBURST									

Table 129. HWTXBUF – TX Buffer Hardware Parameters

Field	Description
TXLC	VUSB_HS_TX_LOCAL_CONTEXT_REGISTERS
(Reserved)	These bits are reserved and should be zero.
TXCHANADD	VUSB_HS_TX_CHAN_ADD
TXADD	VUSB_HS_TX_ADD
TCBURST	VUSB_HS_TX_BURST

9.10.6 HWRXBUF

Address: Base + 014h

Default Value: 0000_05_04h

Attribute: Read Only

Size: 32 bits

RX buffer hardware parameters as defined in [Section 9.7, “System Level Issues and Core Configuration” on page 363](#).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																RXADD							RXBURST								

**Table 130. HWRXBUF – RX Buffer Hardware Parameters**

Field	Description
(Reserved)	These bits are reserved and should be zero.
RXADD	VUSB_HS_RX_ADD
RXBURST	VUSB_HS_RX_BURST

9.11 Host Capability Registers

Host Capability registers specify the software limits, restrictions, and capabilities of the host controller implementation.

9.11.1 CAPLENGTH – EHCI Compliant

Address: Base + 100h

Default Value: 40h

Attribute: Read Only

Size: 8 bits

This register is used to indicate which offset to add to the register base address at the beginning of the Operational Register.

7	6	5	4	3	2	1	0
CAPLENGTH[7:0]							

9.11.2 HCVERSION – EHCI Compliant

Address: Base + 102h

Default Value: 0100h

Attribute: Read Only

Size: 16 bits

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this host controller. The most significant byte of this register represents a major revision and the least significant byte is the minor revision.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HCVERSION[15:0]															

9.11.3 HCSPARAMS – EHCI Compliant with Extensions

Address: Base + 104h

Default Value: 0x0001_0011

Attribute: Read Only

Size: 32 bits

Port steering logic capabilities are described in this register.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)				N_TT				N_PTT				(Rsvd)		PI	N_CC				N_PCC				(Rsvd)		PCP	N_PORTS					

Table 131. HCSPARAMS – Host Control Structural Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
N_TT[3:0]	Number of Transaction Translators (N_TT). This field indicates the number of embedded transaction translators associated with the USB 2.0 host controller. For this implementation, N_TT = "0000." This is a non-EHCI field to support embedded TT.
N_PTT[3:0]	Number of Ports per Transaction Translator (N_PTT). This field indicates the number of ports assigned to each transaction translator within the USB 2.0 host controller. For this implementation, N_PTT = "0000." This is a non-EHCI field to support embedded TT.
PI	Port Indicators (P INDICATOR). This bit indicates whether the ports support port indicator control. When set to one, the port status and control registers include a read/writable field for controlling the state of the port indicator. This field will always be "1."
N_CC[3:0]	Number of Companion Controller (N_CC). This field indicates the number of companion controllers associated with this USB 2.0 host controller. A zero in this field indicates there are no internal Companion Controllers. Port-ownership hand-off is not supported. A value larger than zero in this field indicates there are companion USB1.1 host controller(s). Port-ownership hand-offs are supported. High, Full- and Low-speed devices are supported on the host controller root ports. In this implementation this field will always be "0."
N_PCC[3:0]	Number of Ports per Companion Controller. This field indicates the number of ports supported per internal Companion Controller. It is used to indicate the port routing configuration to the system software. For example, if N_PORTS has a value of 6 and N_CC has a value of 2 then N_PCC could have a value of 3. The convention is that the first N_PCC ports are assumed to be routed to companion controller 1, the next N_PCC ports to companion controller 2, etc. In the previous example, the N_PCC could have been 4, where the first 4 are routed to companion controller 1 and the last two are routed to companion controller 2. The number in this field must be consistent with N_PORTS and N_CC. In this implementation this field will always be "0".
PPC	Port Power Control. This field indicates whether the host controller implementation includes port power control. A one indicates the ports have port power switches. A zero indicates the ports do not have port power switches. The value of this field affects the functionality of the Port Power field in each port status and control register.
N_PORTS[3:0]	Number of downstream ports. This field specifies the number of physical downstream ports implemented on this host controller. The value of this field determines how many port registers are addressable in the Operational Register. Valid values are in the range of 1h to Fh. A zero in this field is undefined. The number of ports for a host implementation can be a parameter from 1 to 8.

9.11.4 HCCPARAMS – EHCI Compliant

Address: Base + 108h

Default Value: 0006h

Attribute: Read Only



Size: 32 bits

This register identifies multiple mode control (time-base bit functionality) addressing capability.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																EECP[7:0]						IST[7:4]				(Rsvd)	ASP	PFL	ADC		

Table 132. HCCPARAMS – Host Control Capability Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
EECP[7:0]	EHCI Extended Capabilities Pointer. Default = 0. This optional field indicates the existence of a capabilities list. A value of 00h indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 40h or greater if implemented to maintain the consistency of the PCI header defined for this class of device. For this implementation this field is always "0".
IST[7:4]	Isochronous Scheduling Threshold. This field indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit [7] is zero, the value of the least significant 3 bits indicates the number of micro-frames a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit [7] is a one, then host software assumes the host controller may cache an isochronous data structure for an entire frame. This field will always be "0".
R	(Reserved). These bits are reserved and should be zero.
ASP	Asynchronous Schedule Park Capability. Default = 1. If this bit is set to a one, then the host controller supports the park feature for high-speed queue heads in the Asynchronous Schedule. The feature can be disabled or enabled and set to a specific level by using the <i>Asynchronous Schedule Park Mode Enable</i> and <i>Asynchronous Schedule Park Mode Count</i> fields in the USBCMD register. This field will always be "1"
PFL	Programmable Frame List Flag. If this bit is set to zero, then the system software must use a frame list length of 1024 elements with this host controller. The USBCMD register Frame List Size field is a read-only register and must be set to zero. If set to a one, then the system software can specify and use a smaller frame list and configure the host controller via the USBCMD register Frame List Size field. The frame list must always be aligned on a 4K-page boundary. This requirement ensures that the frame list is always physically contiguous. This field will always be "1".
ADC	64-bit Addressing Capability. This field will always be "0". No 64-bit addressing capability is supported.

9.11.5 Reserved Register #1

Address: Base + 120h

Default Value: 0001h

Attribute: Read Only - Reserved for Future Use

Size: 16 bits

9.11.6 DCCPARAMS (Non-EHCI)

Address: Base + 124h

Default Value: 0x0000_0184



Attribute: Read Only

Size: 32 bits

These fields describe the overall host capability of the controller.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																							HC	RES	(Rsvd)	(Rsvd)	RES				

Table 133. DCCPARAMS - Device Control Capability Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
HC	Host Capable. When this bit is 1, this controller is capable of operating as an EHCI compatible USB 2.0 host controller.
(Reserved)	This value when read will return as a 1
(Reserved)	These bits are reserved and should be zero.
(Reserved)	This value when read will return as a 4

9.12 Host Operational Registers

Operational registers are comprised of dynamic control or status registers that may be read only, read/write, or read/write to clear. The following sections define the use of these registers.

9.12.1 USBCMD

Address: Base + 140h

Default Value: 0x00080B00 (host mode)

Attribute: Read Only, Read/Write, Write Only (field dependent)

Size: 32 bits

The serial bus host controller executes the command indicated in this register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)								ITC[7:0]								FS2	R	SUTW	ATDTW	ASPE	R	ASP1	ASP0	LR	IAA	ASE	PSE	FS1	FS0	RST	RS



Table 134. USBCMD – USB Command Register (Sheet 1 of 2)

Field	Description																		
R, (Reserved)	These bits are reserved and should be zero.																		
ITC[7:0]	Interrupt Threshold Control —Read / Write. Default 08h. The system software uses this field to set the maximum rate at which the host controller will issue interrupts. ITC contains the maximum interrupt interval measured in micro-frames. Valid values are shown below. <table> <tr> <td>Value</td><td>Maximum Interrupt Interval</td></tr> <tr> <td>00h</td><td>Immediate (no threshold)</td></tr> <tr> <td>01h</td><td>1 micro-frame</td></tr> <tr> <td>02h</td><td>2 micro-frames</td></tr> <tr> <td>04h</td><td>4 micro-frames</td></tr> <tr> <td>08h</td><td>8 micro-frames</td></tr> <tr> <td>10h</td><td>16 micro-frames</td></tr> <tr> <td>20h</td><td>32 micro-frames</td></tr> <tr> <td>40h</td><td>64 micro-frames</td></tr> </table>	Value	Maximum Interrupt Interval	00h	Immediate (no threshold)	01h	1 micro-frame	02h	2 micro-frames	04h	4 micro-frames	08h	8 micro-frames	10h	16 micro-frames	20h	32 micro-frames	40h	64 micro-frames
Value	Maximum Interrupt Interval																		
00h	Immediate (no threshold)																		
01h	1 micro-frame																		
02h	2 micro-frames																		
04h	4 micro-frames																		
08h	8 micro-frames																		
10h	16 micro-frames																		
20h	32 micro-frames																		
40h	64 micro-frames																		
SUTW	Setup TripWire – Read / Write. This bit is used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off, it there exists a hazard when new setup data arrives and the DCD copying setup from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists.																		
ATDTW	Add dTD TripWire – Read / Write. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit must also be cleared by hardware when its state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized.																		
ASPE	Asynchronous Schedule Park Mode Enable (OPTIONAL) — Read / Write Software uses this bit to enable or disable Park mode. When this bit is one, Park mode is enabled. When this bit is a zero, Park mode is disabled. This field is set to “1” in this implementation.																		
ASP[1:0]	Asynchronous Schedule Park Mode Count (OPTIONAL) — Read/Write. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. See “Asynchronous Schedule Park Mode” on page 447 for full operational details. Valid values are 1h to 3h. Software must not write a zero to this bit when <i>Park Mode Enable</i> is a one as this will result in undefined behavior. This field is set to 3h in this implementation.																		
LR	Light Host Controller Reset (OPTIONAL) — Read Only. Not Implemented. This field will always be “0”.																		
IAA	Interrupt on Async Advance Doorbell — Read / Write. This bit is used as a doorbell by software to tell the host controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the host controller has evicted all appropriate cached schedule states, it sets the Interrupt on Async Advance status bit in the USBSTS register. If the Interrupt on Sync Advance Enable bit in the USBINTR register is one, then the host controller will assert an interrupt at the next interrupt threshold. The host controller sets this bit to zero after it has set the Interrupt on Sync Advance status bit in the USBSTS register to one. Software should not write a one to this bit when the asynchronous schedule is inactive. Doing so will yield undefined results. This bit is only used in host mode.																		
ASE	Asynchronous Schedule Enable — Read / Write. Default 0b. This bit controls whether the host controller skips processing the Asynchronous Schedule. Values meaning <table> <tr> <td>0</td><td>Do not process the Asynchronous Schedule.</td></tr> <tr> <td>1</td><td>Use the ASYNCLISTADDR register to access the Asynchronous Schedule.</td></tr> </table> Only the host controller uses this bit.	0	Do not process the Asynchronous Schedule.	1	Use the ASYNCLISTADDR register to access the Asynchronous Schedule.														
0	Do not process the Asynchronous Schedule.																		
1	Use the ASYNCLISTADDR register to access the Asynchronous Schedule.																		

**Table 134. USBCMD – USB Command Register (Sheet 2 of 2)**

Field	Description
PSE	Periodic Schedule Enable— Read / Write. Default 0b. This bit controls whether the host controller skips processing the Periodic Schedule. Values meaning 0 Do not process the Periodic Schedule 1 Use the PERIODICLISTBASE register to access the Periodic Schedule. Only the host controller uses this bit.
FS[2:0]	Frame List Size — (Read/Write or Read Only). Default 000b. This field is Read/Write only if Programmable Frame List Flag in the HCCPARAMS registers is set to one. This field specifies the size of the frame list that controls which bits in the Frame Index Register should be used for the Frame List Current index. Note that this field is made up from USBCMD bits 15, 3 and 2. Values meaning 000 1024 elements (4096 bytes) Default value 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes) Only the host controller uses this field.
RST	Controller Reset (RESET) — Read / Write. Software uses this bit to reset the controller. This bit is set to zero by the Host Controller when the reset process is complete. Software cannot terminate the reset process early by writing a zero to this register. Host Controller: When software writes a one to this bit, the Host Controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit to a one when the HCHalted bit in the USBSTS register is a zero. Attempting to reset an actively running host controller will result in undefined behavior.
RS	Run/Stop (RS) – Read/Write. Default 0b. 0 = Stop. 1 = Run. Host Controller: When set to a 1, the Host Controller proceeds with the execution of the schedule. The Host Controller continues execution as long as this bit is set to a one. When this bit is set to 0, the Host Controller completes the current transaction on the USB and then halts. The HC Halted bit in the status register indicates when the Host Controller has finished the transaction and has entered the stopped state. Software should not write a one to this field unless the host controller is in the Halted state (i.e. HCHalted in the USBSTS register is a one).

9.12.2 USBSTS

Address: Base + 144h

Default Value: 0x00001000 (host mode)

Attribute: Read Only, Read/Write, Read/Write-Clear (field dependent)

Size: 32 bits

This register indicates various states of the Host Controller and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																AS	PS	RCL	HCH	(Rsvd)	(Rsvd)	(Rsvd)	(Rsvd)	SRI	(Rsvd)	AAI	SEI	FRI	PCI	UEI	UI

Table 135. USBSTS – USB Status (Sheet 1 of 2)

Field	Description
(Reserved)	These bits are reserved and should be zero.
AS	Asynchronous Schedule Status — Read Only. 0=Default. This bit reports the current real status of the Asynchronous Schedule. When set to zero the asynchronous schedule status is disabled and if set to one the status is enabled. The Host Controller is not required to immediately disable or enable the Asynchronous Schedule when software transitions the Asynchronous Schedule Enable bit in the USBCMD register. When this bit and the Asynchronous Schedule Enable bit are the same value, the Asynchronous Schedule is either enabled (1) or disabled (0). Only used by the host controller.
PS	Periodic Schedule Status — Read Only. 0=Default. This bit reports the current real status of the Periodic Schedule. When set to zero the periodic schedule is disabled, and if set to one the status is enabled. The Host Controller is not required to immediately disable or enable the Periodic Schedule when software transitions the Periodic Schedule Enable bit in the USBCMD register. When this bit and the Periodic Schedule Enable bit are the same value, the Periodic Schedule is either enabled (1) or disabled (0). Only used by the host controller.
RCL	Reclamation — Read Only. 0=Default. This is a read-only status bit used to detect an empty asynchronous schedule. Only used by the host controller.
HCH	HCHalted — Read Only. 1=Default. This bit is a zero whenever the Run/Stop bit is a one. The Host Controller sets this bit to one after it has stopped executing because of the Run/Stop bit being set to 0, either by software or by the Host Controller hardware (e.g. internal error). Only used by the host controller.
R	(Reserved). These bits are reserved and should be zero.
(Reserved)	(Reserved) - These bits are reserved and should be zero
SRI	SOF Received – R/WC. 0=Default. In host mode, this bit will be set every 125us and can be used by host controller driver as a time base. Software writes a 1 to this bit to clear it. This is a non-EHCI status bit.
(Reserved)	(Reserved) - These bits are reserved and should be zero
AAI	Interrupt on Async Advance — R/WC. 0=Default. System software can force the host controller to issue an interrupt the next time the host controller advances the asynchronous schedule by writing a one to the Interrupt on Async Advance Doorbell bit in the USBCMD register. This status bit indicates the assertion of that interrupt source. Only used by the host controller.
SEI	System Error— R/WC. This bit is not used in this implementation and will always be set to “0”.
FRI	Frame List Rollover — R/WC. The Host Controller sets this bit to a one when the Frame List Index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the Host Controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host controller.

**Table 135. USBSTS – USB Status (Sheet 2 of 2)**

Field	Description
PCI	Port Change Detect — R/WC. The Host Controller sets this bit to a one when on any port a Connect Status occurs, a Port Enable/Disable Change occurs, or the Force Port Resume bit is set as the result of a J-K transition on the suspended port. This bit is not EHCI compatible.
UEI	USB Error Interrupt (USBERRINT) — R/WC. When completion of a USB transaction results in an error condition, this bit is set by the Host Controller. This bit is set along with the USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set See Section (Reference Host Operation Model: Transfer/Transaction Based Interrupt – i.e. 4.15.1 in EHCI) for a complete list of host error interrupt conditions.
UI	USB Interrupt (USBINT) — R/WC. This bit is set by the Host Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the Host Controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

9.12.3 USBINTR

Address: Base + 148h

Default Value: 0x00000000

Attribute: Read/Write

Size: 32 bits

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB Status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																							(Rsvd)	(Rsvd)	(Rsvd)	AAE	SEE	FRE	PCE	UEE	UE

Table 136. USBINTR – USB Interrupt Enable

Field	Interrupt Source	Description
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
AAE	Interrupt on Async Advance Enable	When this bit is a one, and the Interrupt on Async Advance bit in the USBSTS register is a one, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the Interrupt on Async Advance bit. Only used by the host controller.
SEE	System Error Enable	When this bit is a one, and the System Error bit in the USBSTS register is a one, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the System Error bit.



Table 136. USBINTR – USB Interrupt Enable

Field	Interrupt Source	Description
FRE	Frame List Rollover Enable	When this bit is a one, and the <i>Frame List Rollover</i> bit in the USBSTS register is a one, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the <i>Frame List Rollover</i> bit. Only used by the host controller.
PCE	Port Change Detect Enable	When this bit is a one, and the <i>Port Change Detect</i> bit in the USBSTS register is a one, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the <i>Port Change Detect</i> bit.
UEE	USB Error Interrupt Enable	When this bit is a one, and the <i>USBERRINT</i> bit in the USBSTS register is a one, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the <i>USBERRINT</i> bit in the USBSTS register.
UE	USB Interrupt Enable	When this bit is a one, and the <i>USBINT</i> bit in the USBSTS register is a one, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the <i>USBINT</i> bit.

9.12.4 FRINDEX

Address: Base + 14Ch

Default Value: Undefined (free running counter)

Attribute: Read/Write in host mode

Size: 32 bits

This register is used by the host controller to index the periodic frame list. The register updates every 125 ms (once each micro-frame). Bits [N: 3] are used to select a particular entry in the Periodic Frame List during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the Frame List Size field in the USBCMD register.

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the Host Controller is in the 'Halted' state as indicated by the HCHalted bit. A write to this register while the Run/Stop bit is set to a one produces undefined results. Writes to this register also affect the SOF value.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																		FRINDEX[13:0]													

**Table 137. FRINDEX – USB Frame Index**

Field	Description																		
(Reserved)	These bits are reserved and should be zero.																		
FRINDEX	Frame Index. The value, in this register, increments at the end of each time frame (e.g. micro-frame). Bits [N: 3] are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or micro-frames) before moving to the next index. The following illustrates values of N based on the value of the Frame List Size field in the USBCMD register, when used in host mode. <table> <tr> <td>USBCMD (Frame List Size)</td><td>Number Elements N</td></tr> <tr> <td>000b (1024)</td><td>12</td></tr> <tr> <td>001b (512)</td><td>11</td></tr> <tr> <td>010b (256)</td><td>10</td></tr> <tr> <td>011b (128)</td><td>9</td></tr> <tr> <td>100b (64)</td><td>8</td></tr> <tr> <td>101b (32)</td><td>7</td></tr> <tr> <td>110b (16)</td><td>6</td></tr> <tr> <td>111b (8)</td><td>5</td></tr> </table> bits 2:0 indicate the current microframe.	USBCMD (Frame List Size)	Number Elements N	000b (1024)	12	001b (512)	11	010b (256)	10	011b (128)	9	100b (64)	8	101b (32)	7	110b (16)	6	111b (8)	5
USBCMD (Frame List Size)	Number Elements N																		
000b (1024)	12																		
001b (512)	11																		
010b (256)	10																		
011b (128)	9																		
100b (64)	8																		
101b (32)	7																		
110b (16)	6																		
111b (8)	5																		

9.12.5 CTRLDSSEGMENT

Address: Base + 150h

Default Value: 0x00000000

Attribute: Read Only

Size: 32 bits

This register is not used in this implementation.

9.12.6 PERIODICLISTBASE

Address: Base + 154h

Default Value: 0x00000000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

9.12.6.1 Host Controller (PERIODICLISTBASE)

This 32-bit register contains the beginning address of the Periodic Frame List in the system memory. HCD loads this register prior to starting the schedule execution by the Host Controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the Frame Index Register (FRINDEX) to enable the Host Controller to step through the Periodic Frame List in sequence.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PERBASE[31:12]																				(Reserved)											

**Table 138. PERIODICLISTBASE - Host Controller Frame List Base Address**

Field	Description
BASEADR	Base Address (Low). These bits correspond to memory address signals [31:12], respectively. Only used by the host controller.
(Reserved)	Must be written as zeros. During runtime, the values of these bits are undefined.

9.12.7 ASYNCLISTADDR; ENDPOINTLISTADDR

Address: Base + 158h

Default Value: 0x00000000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

9.12.7.1 Host Controller (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits [4:0] of this register cannot be modified by the system software and will always return a zero when read.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASYBASE[31:5]																									(Reserved)						

Table 139. ASYNCLISTADDR - Host Controller Next Asynchronous Address

Field	Description
ASYBASE[31:5]	Link Pointer Low (LPL). These bits correspond to memory address signals [31:5], respectively. This field may only reference a Queue Head (OH). Only used by the host controller.
(Reserved)	These bits are reserved and their value has no effect on operation.

9.12.8 BURSTSIZE

Address: Base + 160h

Default Value: 0x0000_0404

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

This register is used to control dynamically change the burst size used during data movement on the initiator (master) interface.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)															TXPBURST								RXPBURST								

**Table 140. BURSTSIZE - Host Controller Embedded TT Async. Buffer Status**

Field	Description
(Reserved)	These bits are reserved and their value has no effect on operation.
TXPBURST	Programmable TX Burst Length. (Read / Write) Default is the constant VUSB_HS_TX_BURST. This register represents the maximum length of a the burst in 32-bit words while moving data from system memory to the USB bus.
RXPBURST	Programmable RX Burst Length. (Read / Write) Default is the constant VUSB_HS_RX_BURST. This register represents the maximum length of a the burst in 32-bit words while moving data from the USB bus to system memory.

9.12.9 TXFILLTUNING

Address: Base + 164h

Default Value: 0x00020000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)										TXFIFOTHRES					(Rsvd)			TXSCHEALTH					TXSCHOH								

The fields in this register control performance tuning associated with how the host controller posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

- T_0 = Standard packet overhead
- T_1 = Time to send data payload
- T_{ff} = Time to fetch packet into TX FIFO up to specified level.
- T_s = Total Packet Flight Time (send-only) packet
- $T_s = T_0 + T_1$
- T_p = Total Packet Time (fetch and send) packet
- $T_p = T_{ff} + T_0 + T_1$

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure T_p remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at anytime during the pre-fill operation the time remaining the [micro]frame is $< T_s$ then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the host controller will eventually recover, a mark will be made the scheduler health counter to note the occurrence of a “back-off” event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHEALTH (T_{ff}) described below.

9.12.10 CONFIGFLAG

Address: Base + 180h



Default Value: 0x00000001

Attribute: Read Only

Size: 32 bits

This register is not used in this implementation. A read from this register returns a constant of a 00000001h to indicate that all port routings default to this host controller.

9.12.11 PORTSCx

Address: Base + 184h + (4*(Port Number – 1))

where Port Number = 1,2,3,...8

Default Value: IIII_RR00_0000_0000_0000_XX00_0000_0000b (host mode)

I = Implementation dependent, this implementation: 0x4

X = Unknown

RR = after reset, the PSPD field is set to 0b11, which is not a defined speed.

Attribute: R0, Read/Write, R/WC (field dependent)

Size: 32 bits

9.12.11.1 Host Controller

A host controller must implement one to eight port registers. The number of port registers implemented by a particular instantiation of a host controller is documented in the HCSPARAMs register. Software uses this information as an input parameter to determine how many ports need service.

This register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTS	STS	PTW	(Rsvd)	PSPD		(Rsvd)	PFSC	PHCD	WKOC	WKDS	WKCEN	PTC[3:0]				PIC		PO	PS	LS		HSP	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS



Table 141. PORTSCx - Port Status Control[1:8] (Sheet 1 of 4)

Field	Description
PTS	Parallel Transceiver Select – Read / Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY_MODE to control which parallel transceiver interface is selected. If VUSB_HS_PHY_MODE is set for 0,1 or 2 then this bit is read only. If VUSB_HS_PHY_MODE is 3,4 or 5 then this bit is read/write. This bit is reset to 1 if VUSB_HS_PHY_MODE selects a default interface of Philips*, else it is reset to 0. Writing this bit to 0 selects the UTMI parallel Interface. Writing to 1 selects the Philips parallel Interface. This bit is not defined in the EHCI specification.
STS	Serial Transceiver Select – Read / Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY_MODE to control whether the parallel or serial transceiver interface is selected. If VUSB_HS_PHY_MODE is set for 0,1 or 2 then this bit is read only. If VUSB_HS_PHY_MODE is 3,4 or 5 then this bit is read/write. This bit is reset to 1 if VUSB_HS_PHY_MODE selects a default interface of Serial FS else it is reset to 0. Writing this bit to 0 selects the parallel transceiver Interface. Writing this bit to 1 selects the Serial (Full Speed) transceiver interface. For the UTMI interface, this bit can be used to select between a Parallel UTMI PHY and a standard 1.1 PHY. For the Philips interface, this bit is used to prevent use of the Parallel PHY and generate signalling exclusively through the serial interface. (For Philips this also implies no chirping.) This bit is not defined in the EHCI specification.
PTW	Parallel Transceiver Width – Read / Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY8_16 to control whether the data bus width of the UTMI transceiver interface. If VUSB_HS_PHY8_16 is set for 0 or 1 then this bit is read only. If VUSB_HS_PHY8_16 is 2 or 3 then this bit is read/write. This bit is reset to 1 if VUSB_HS_PHY8_16 selects a default UTMI interface width of 16-bits else it is reset to 0. Writing this bit to 0 selects the 8-bit [60MHz] UTMI interface. Writing this bit to 1 selects the 16-bit [30MHz] UTMI interface. This bit is not defined in the EHCI specification.
PSPD	Port Speed – Read Only. This register field indicates the speed at which the port is operating. For HS mode operation in the host controller the port routing steers data to the Protocol engine. For FS and LS mode operation in the host controller, the port routing steers data to the Protocol Engine w/ Embedded Transaction Translator. This implementation does not support HS mode of operation. 00 – Full Speed 01 – Low Speed 10 – High Speed (Not supported) This bit is not defined in the EHCI specification.
PFSC	Port Force Full Speed Connect – Read / Write. Default = 0b. Writing this bit to a 1b will force the port to only connect at Full Speed. It disables the chirp sequence that allows the port to identify itself as High Speed. This is useful for testing FS configurations with a HS host, hub or device. This bit is not defined in the EHCI specification. This bit is for debugging purposes.
PHCD	PHY Low Power Suspend - Clock Disable (PLPSCD) – Read / Write. Default = 0b. Writing this bit to a 1b will disable the PHY clock. Writing a 0b enables it. Reading this bit will indicate the status of the PHY clock. NOTE: The PHY clock cannot be disabled if it is being used as the system clock. In host mode, the PHY can be put into Low Power Suspend – Clock Disable when the downstream device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of software. This bit is not defined in the EHCI specification.
WKOC	Wake on Over-current Enable (WKOC_E) – Read / Write. Default = 0b. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events. This field is zero if Port Power() is zero. This bit is output from the controller as signal pwrctl_wake_ovcurr_en (host core only) for use by an external power control circuit.



Table 141. PORTSCx - Port Status Control[1:8] (Sheet 2 of 4)

Field	Description																
WKDC	Wake on Disconnect Enable (WKDSCNNT_E) — Read / Write. Default=0b. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events. This field is zero if <i>Port Power()</i> is zero. This bit is output from the controller as signal pwrctl_wake_dscnnt_en (host core only) for use by an external power control circuit.																
WKN	Wake on Connect Enable (WKCNT_E) — Read / Write. Default=0b. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events. This field is zero if <i>Port Power()</i> is zero. This bit is output from the controller as signal pwrctl_wake_dscnnt_en (host core only) for use by an external power control circuit.																
PTC[3:0]	Port Test Control — Read / Write. Default = 0000b. Any other value than zero indicates that the port is operating in test mode. <table> <tr> <th>Value</th><th>Specific Test</th></tr> <tr> <td>0000b</td><td>Not enabled</td></tr> <tr> <td>0001b</td><td>J_STATE</td></tr> <tr> <td>0010b</td><td>K_STATE</td></tr> <tr> <td>0011b</td><td>SEQ_NAK</td></tr> <tr> <td>0100b</td><td>Packet</td></tr> <tr> <td>0101b</td><td>FORCE_ENABLE</td></tr> <tr> <td>0110b to 1111b</td><td>(Reserved)</td></tr> </table> Refer to Chapter 7 of the USB Specification Revision 2.0 for details on each test mode.	Value	Specific Test	0000b	Not enabled	0001b	J_STATE	0010b	K_STATE	0011b	SEQ_NAK	0100b	Packet	0101b	FORCE_ENABLE	0110b to 1111b	(Reserved)
Value	Specific Test																
0000b	Not enabled																
0001b	J_STATE																
0010b	K_STATE																
0011b	SEQ_NAK																
0100b	Packet																
0101b	FORCE_ENABLE																
0110b to 1111b	(Reserved)																
PIC[1:0]	Port Indicator Control — Read / Write. Default = 0b. <table> <tr> <th>Bit Value</th><th>Meaning</th></tr> <tr> <td>00b</td><td>Port indicators are off</td></tr> <tr> <td>01b</td><td>Amber</td></tr> <tr> <td>10b</td><td>Green</td></tr> <tr> <td>11b</td><td>Undefined</td></tr> </table> Refer to the USB Specification Revision 2.0 for a description on how these bits are to be used. This field is output from the controller as signals port_ind_ctl_1 & port_ind_ctl_0 for use by an external led driving circuit.	Bit Value	Meaning	00b	Port indicators are off	01b	Amber	10b	Green	11b	Undefined						
Bit Value	Meaning																
00b	Port indicators are off																
01b	Amber																
10b	Green																
11b	Undefined																
PO	Port Owner—Read / Write. Default = 0. This bit unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is zero. System software uses this field to release ownership of the port to a selected host controller (in the event that the attached device is not a high-speed device). Software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port. Port owner handoff is not implemented in this design, therefore this bit will always be 0.																
PP	Port Power (PP)—Read / Write 1b1b/0b – RW. Host controller requires port power control switches. This bit represents the current setting of the switch (0=off, 1=on). When power is not available on a port (i.e. PP equals a 0), the port is non-functional and will not report attaches, detaches, etc. When an over-current condition is detected on a powered port and PPC is a one, the PP bit in each affected port may be transitioned by the host controller driver from a one to a zero (removing power from the port). This feature is implemented in the host controller (PPC = 1).																
LS[1:0]	Line Status—Read Only. These bits reflect the current logical levels of the D+ (bit 11) and D- (bit 10) signal lines. The encoding of the bits are: <table> <tr> <th>Bits [11:10]</th><th>Meaning</th></tr> <tr> <td>00b</td><td>SE0</td></tr> <tr> <td>10b</td><td>J-state</td></tr> <tr> <td>01b</td><td>K-state</td></tr> <tr> <td>11b</td><td>Undefined</td></tr> </table> In host mode, the use of line state by the host controller driver is not necessary (unlike EHCI), because the port controller state machine and the port routing manage the connection of LS and FS.	Bits [11:10]	Meaning	00b	SE0	10b	J-state	01b	K-state	11b	Undefined						
Bits [11:10]	Meaning																
00b	SE0																
10b	J-state																
01b	K-state																
11b	Undefined																



Table 141. PORTSCx - Port Status Control[1:8] (Sheet 3 of 4)

Field	Description								
HSP	High-Speed Port — Read Only. Default = 0b. When the bit is one, the host connected to the port is in high-speed mode and if set to zero, the host connected to the port is not in a high-speed mode. Note: HSP is redundant with PSPD(27:26) but will remain in the design for compatibility. This bit is not defined in the EHCI specification. This implementation does not support HS mode of operation.								
PR	Port Reset This field is zero if <i>Port Power()</i> is zero. In Host Mode: Read/Write. 0 = Port is not in Reset. 1 = Port is in Reset. Default 0. When software writes a one to this bit the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. <i>This bit will automatically change to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver.</i>								
SUSP	Suspend In Host Mode: Read/Write. 0 = Port not in suspend state. 1 = Port in suspend state. Default=0. Port Enabled Bit and Suspend bit of this register define the port states as follows: <table border="0"> <tr> <td>Bits [Port Enabled, Suspend]</td> <td>Port State</td> </tr> <tr> <td>0x</td> <td>Disable</td> </tr> <tr> <td>10</td> <td>Enable</td> </tr> <tr> <td>11</td> <td>Suspend</td> </tr> </table> When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB. The host controller will unconditionally set this bit to zero when software sets the <i>Force Port Resume</i> bit to zero. A write of zero to this bit is ignored by the host controller. If host software sets this bit to a one when the port is not enabled (i.e. <i>Port enabled</i> bit is a zero) the results are undefined. This field is zero if <i>Port Power()</i> is zero in host mode.	Bits [Port Enabled, Suspend]	Port State	0x	Disable	10	Enable	11	Suspend
Bits [Port Enabled, Suspend]	Port State								
0x	Disable								
10	Enable								
11	Suspend								
FPR	Force Port Resume —Read/Write. 0 = No resume (K-state) detected/driven on port. 1 = Resume detected/driven on port. Default = 0. In Host Mode: Software sets this bit to one to drive resume signaling. The Host Controller sets this bit to one if a J-to-K transition is detected while the port is in the Suspend state. When this bit transitions to a one because a J-to-K transition is detected, the <i>Port Change Detect</i> bit in the USBSTS register is also set to one. <i>This bit will automatically change to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver.</i> Note that when the Host controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a one. This bit will remain a one until the port has switched to the high-speed idle. Writing a zero has no affect because the port controller will time the resume operation clear the bit the port control state switches to HS or FS idle. This field is zero if <i>Port Power()</i> is zero in host mode. This bit is not-EHCI compatible.								



Table 141. PORTSCx - Port Status Control[1:8] (Sheet 4 of 4)

Field	Description
OCC	Over-current Change—R / WC. Default=0. 1=This bit gets set to one when there is a change to Over-current Active. Software clears this bit by writing a one to this bit position. For host implementations the user can provide over-current detection to the vbus_pwr_fault input for this condition.
OCA	Over-current Active—Read Only. Default 0. 0 = This port does not have an over-current condition. 1 = This port currently has an over-current condition. This bit will automatically transition from one to zero when the over current condition is removed. For host implementations the user can provide over-current detection to the vbus_pwr_fault input for this condition.
PEC	Port Enable/Disable Change—R/WC. 0 = No change. 1 = Port enabled/disabled status has changed. Default = 0. In Host Mode: For the root hub, this bit gets set to a one only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a one to it. This field is zero if <i>Port Power()</i> is zero.
PE	Port Enabled/Disabled—Read/Write. 0 = Disable. 1 = Enable. Default 0. In Host Mode: Ports can only be enabled by the host controller as a part of the reset and enable. Software cannot enable a port by writing a one to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. Note that the bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host controller and bus events. When the port is disabled, (0b) downstream propagation of data is blocked except for reset. This field is zero if <i>Port Power()</i> is zero in host mode.
CSC	Connect Status Change—R/WC. 0 = No change. 1 = Change in Current Connect Status. Default 0. In Host Mode: Indicates a change has occurred in the port's Current Connect Status. The host controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it. This field is zero if <i>Port Power()</i> is zero in host mode.
CCS	Current Connect Status—Read Only. In Host Mode: 0 = No device is present. 1 = Device is present on port. Default = 0. This value reflects the current state of the port, and may not correspond directly to the event that caused the <i>Connect Status Change</i> bit (Bit 1) to be set. This field is zero if <i>Port Power()</i> is zero in host mode.



9.12.12 USBMODE

Address: Base + 1A8h

Default Value: 0x00000000

00000003h (host mode)

Attribute: R/WO, Read Only

Size: 32 bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																										SDI	SLOW	ES	CM		

Table 142. USBMODE - USB Device Mode

Field	Description
(Reserved)	These bits are reserved and should be zero.
SDIS	Stream Disable Mode. (0 – Inactive [default]; 1 – Active) <u>Host Mode</u> : Setting to a '1' ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring that the TX latency is filled to capacity before the packet is launched onto the USB. Note: Time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature. Note: The use of this feature substantially limits of the overall USB performance that can be achieved.
(Reserved)	(Reserved)
ES	Endian Select – Read / Write. This bit can change the byte alignment of the transfer buffers to match the host microprocessor. The bit fields in the microprocessor interface and the data structures are unaffected by the value of this bit because they are based upon the 32-bit word. Bit Meaning 0 little-endian [Default] 1 big-endian
CM[1:0]	Controller Mode – R / WO. Controller mode is defaulted to the proper mode for host only implementations. For those designs that contain host capability, the controller will default to an idle state and will need to be initialized to the desired operating mode after reset. For host controllers, this register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to the <i>RESET</i> bit in the USB_CMD register before reprogramming this register. Bit Meaning 00 Idle [Default for combination host] 01 (Reserved) 10 (Reserved) 11 Host Controller [Default for host only controller]

9.13 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the Enhanced Host Controller (hardware). The data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of a Periodic Schedule, Periodic Frame List, Asynchronous Schedule, Isochronous Transaction Descriptors, Split-transaction Isochronous Transfer Descriptors, Queue Heads, and Queue Element Transfer Descriptors.



The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using Isochronous Transaction Descriptors. Isochronous split-transaction data streams are managed with Split-transaction Isochronous Transfer Descriptors. All Interrupt, Control, and Bulk data streams are managed via queue heads and Queue Element Transfer Descriptors¹. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note that software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

The data structures defined in this section are (from the host controller's perspective) a mix of read-only and read/writable fields. The host controller must preserve the read-only fields on all data structure writes.

9.13.1 Periodic Frame List

This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the *PERIODICLISTBASE* address register and the *FRINDEX* register. The periodic schedule is based on an array of pointers called the Periodic Frame List. The *PERIODICLISTBASE* address register is combined with the *FRINDEX* register to produce a memory pointer into the frame list. The Periodic Frame List implements a *sliding window* of work over time as shown in Figure 45. (This figure is identical to Figure 38 on page 356; it is duplicated here for convenience.)

The periodic frame list is a 4K page aligned array of Frame List Link pointers. The length of the frame list may be programmable. The programmability of the periodic frame list is exported to system software via the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, the length can be selected by system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (i.e. the number of elements) is accomplished by system software writing the appropriate value into *Frame List Size* field in the USBCMD register.

Frame List Link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the Frame List.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Fame List Link Pointer																												0	Typ		03-00H	

Frame List Link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least significant bits in a frame list pointer are used to key the host controller as to the type of object the pointer is referencing.

1. Split transaction Interrupt, Bulk and Control are also managed using queue heads and queue element transfer descriptors.

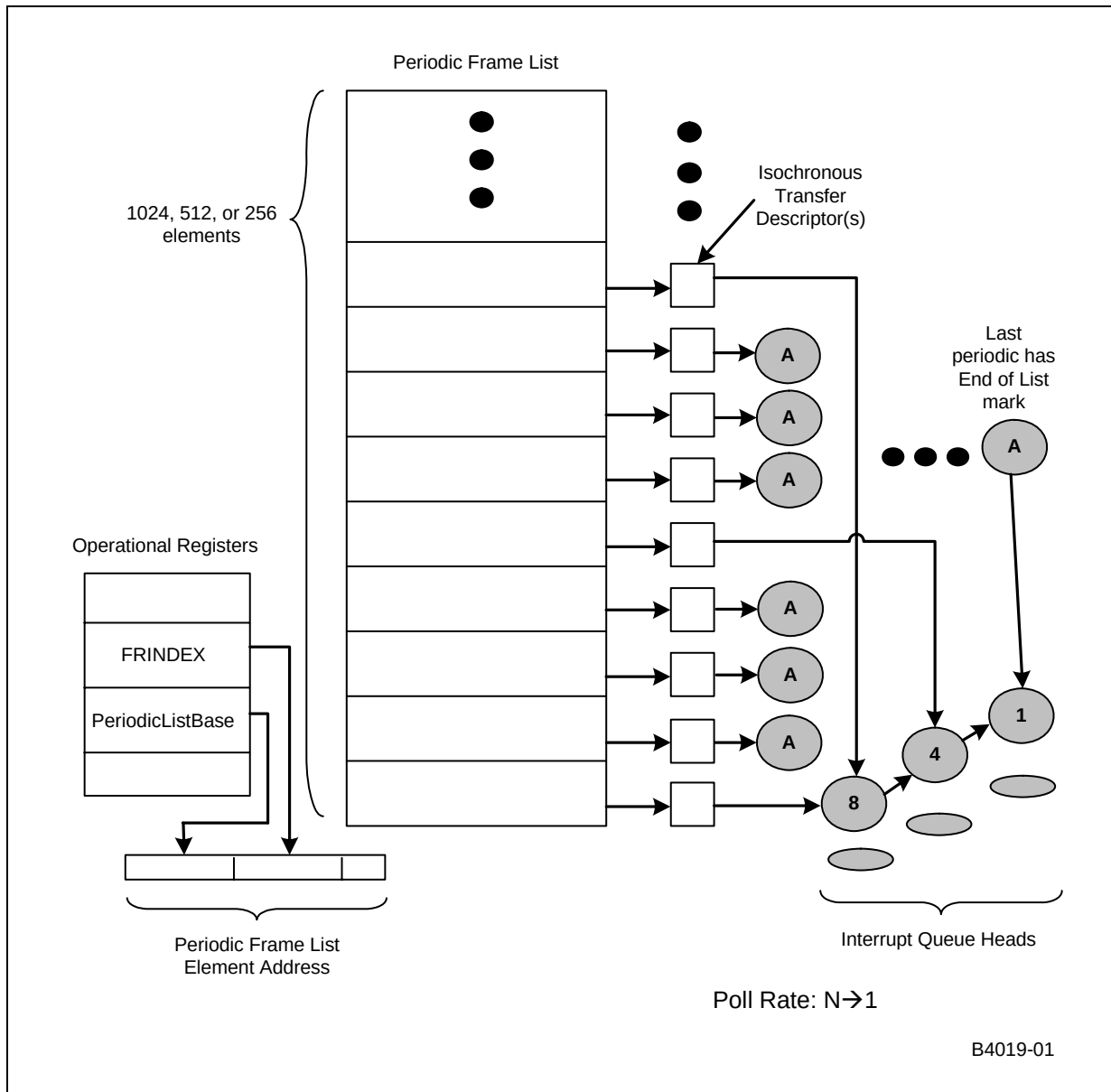


The least significant bit is the *T-Bit* (bit 0). When this bit is set to a one, the host controller will never use the value of the frame list pointer as a physical memory pointer. The *Typ* field is used to indicate the exact type of data structure being referenced by this pointer. The value encodings are:

Table 143. Typ Field Value Definitions

Value	Meaning
00b	Isochronous Transfer Descriptor
01b	Queue Head
10b	Split Transaction Isochronous Transfer Descriptor.
11b	Frame Span Traversal Node.

Figure 45. Periodic Schedule Organization

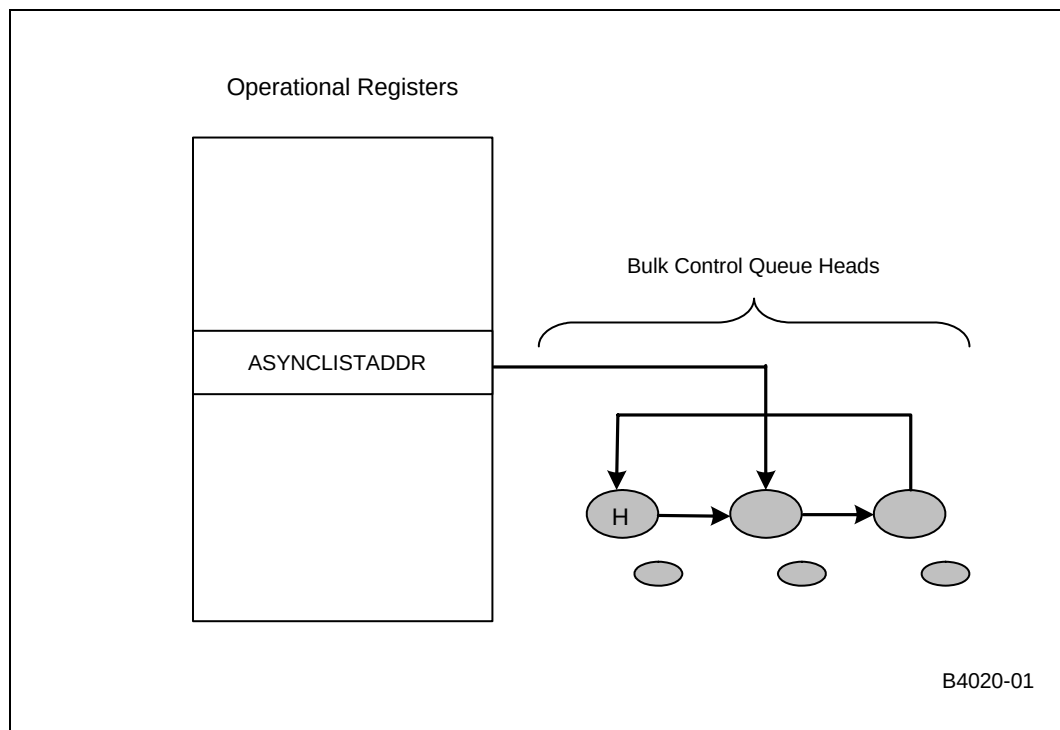


9.13.2 Asynchronous List Queue Head Pointer

The Asynchronous Transfer List (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers use this list only when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.



Figure 46. Asynchronous Schedule Organization



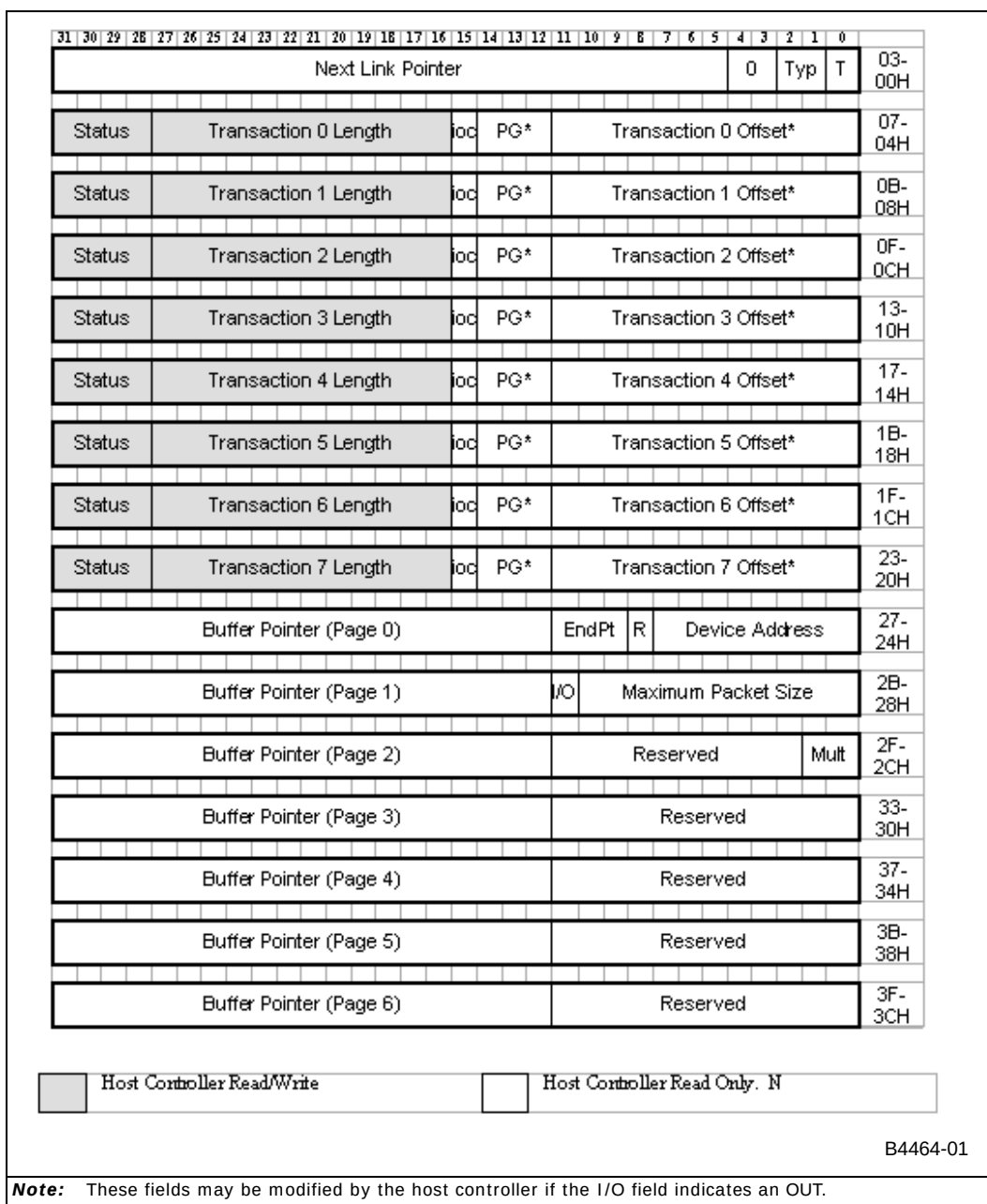
The Asynchronous list is a simple circular list of queue heads as shown in [Figure 46](#). (This figure is identical to [Figure 39 on page 357](#); it is duplicated here for convenience.) The ASYNCLISTADDR register is simply pointer to the *next* queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

9.13.3 Isochronous (High-Speed) Transfer Descriptor (iTd)

The format of an isochronous transfer descriptor is illustrated in "[Isochronous Transaction Descriptor \(iTd\)](#)" on [page 392](#). This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.



Figure 47. Isochronous Transaction Descriptor (iTD)



9.13.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

**Table 144. Next Schedule Element Pointer**

Bit	Description
31:5	Link Pointer (LP). These bits correspond to memory address signals [31:5], respectively. This field points to another Isochronous Transaction Descriptor (iTD/siTD) or Queue Head (QH).
4:3	(Reserved). These bits are reserved and their value has no effect on operation. Software should initialize this field to zero.
2:1	QH / (s)iTD Select (Typ). This field indicates to the Host Controller whether the item referenced is an iTD, siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor) 11b FSTN (frame span traversal node)
0	Terminate (T). 0 = Link Pointer field is valid. 1 = Link Pointer field is not valid.

9.13.3.2 iTD Transaction Status and Control List

DWords 1 through 8 are eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The *PG* and *Transaction X Offset* fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description plus the endpoint information contained in the first three DWords of the Buffer Page Pointer list, to execute a transaction on the USB.

Table 145. iTD Transaction Status and Control (Sheet 1 of 2)

Bit	Description
31:28	Status. This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:
31	Active. Set to one by software to enable the execution of an isochronous transaction by the Host Controller. When the transaction associated with this descriptor is completed, the Host Controller sets this bit to zero indicating that a transaction for this element should not be executed when it is next encountered in the schedule.
30	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, no action is necessary.
29	Babble Detected. Set to one by the Host Controller during status update when "babble" is detected during the transaction generated by this descriptor.
28	Transaction Error (XactErr). Set to one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, etc.). This bit may only be set for isochronous IN transactions.
27:16	Transaction X Length. For an OUT, this field is the number of data bytes the host controller will send during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (e.g. 0àzero length data, 1àone byte, 2àtwo bytes, etc.). The maximum value this field may contain is 0xC00 (3072).

**Table 145. iTD Transaction Status and Control (Sheet 2 of 2)**

Bit	Description
15	Interrupt On Complete (IOC). If this bit is set to one, it specifies that when this transaction completes, the Host Controller should issue an interrupt at the next interrupt threshold.
14:12	Page Select (PG). These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11:0	Transaction X Offset. This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

9.13.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9-15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow for 3 (transactions) * 1,024 (maximum packet size) * 8 (transaction records) (24,576 bytes) to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4K-aligned page pointer, the least significant 12 bits in several of the page pointers are used for other purposes.

Table 146. iTD Buffer Pointer Page 0 (Plus)

Bit	Description
31:12	Buffer Pointer (Page 0). This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	(Reserved). Bit reserved for future use and should be initialized by software to zero.
6:0	Device Address. This field selects the specific device serving as the data source or sink.

Table 147. iTD Buffer Pointer Page 1 (Plus)

Bit	Description
31:12	Buffer Pointer (Page 1). This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11	Direction (I/O). 0 = OUT 1 = IN. This field encodes whether the high-speed transaction should use an IN or OUT PID.
10:0	Maximum Packet Size. This directly corresponds to the maximum packet size of the associated endpoint (<i>wMaxPacketSize</i>). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (.e.g. per micro-frame). This field is used with the <i>Multi</i> field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (400h). Any value larger yields undefined results.


Table 148. iTD Buffer Pointer Page 2 (Plus)

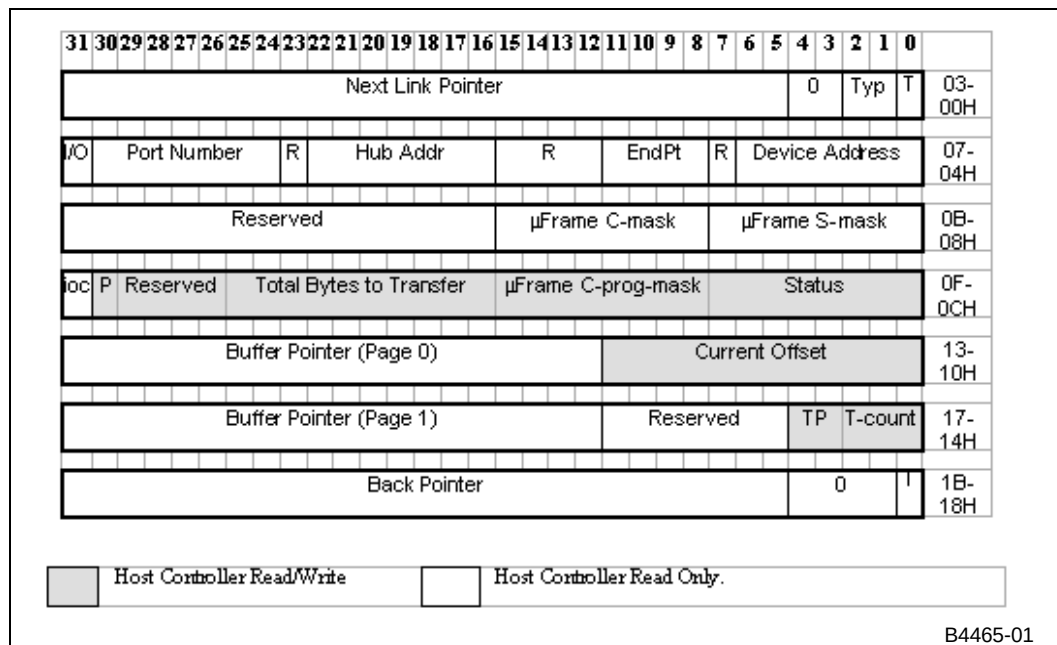
Bit	Description
31:12	Buffer Pointer. This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:2	(Reserved). This bit reserved for future use and should be zero.
1:0	Multi. This field is used to indicate to the host controller the number of transactions that should be executed per transaction description (e.g. per micro-frame). The valid values are: Value Meaning 00b (Reserved). A zero in this field yields undefined results. 01b One transaction to be issued for this endpoint per micro-frame 10b Two transactions to be issued for this endpoint per micro-frame 11b Three transactions to be issued for this endpoint per micro-frame

Table 149. iTD Buffer Pointer Page 3-6

Bit	Description
31:12	Buffer Pointer. This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:0	(Reserved). These bits reserved for future use and should be zero.

9.13.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

Figure 48. Split-transaction Isochronous Transaction Descriptor (siTD)


9.13.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.



Table 150. Next Link Pointer

Bit	Description										
31:5	Next Link Pointer (LP). This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.										
4:3	(Reserved). These bits must be written as zeros.										
2:1	QH / (s)iTD Select (Typ). This field indicates to the Host Controller whether the item referenced is an iTD/siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: <table><tr><td>Value</td><td>Meaning</td></tr><tr><td>00b</td><td>iTD (isochronous transfer descriptor)</td></tr><tr><td>01b</td><td>QH (queue head)</td></tr><tr><td>10b</td><td>siTD (split transaction isochronous transfer descriptor)</td></tr><tr><td>11b</td><td>FSTN (frame span traversal node)</td></tr></table>	Value	Meaning	00b	iTD (isochronous transfer descriptor)	01b	QH (queue head)	10b	siTD (split transaction isochronous transfer descriptor)	11b	FSTN (frame span traversal node)
Value	Meaning										
00b	iTD (isochronous transfer descriptor)										
01b	QH (queue head)										
10b	siTD (split transaction isochronous transfer descriptor)										
11b	FSTN (frame span traversal node)										
0	Terminate (T). 0 = Link Pointer field is valid. 1 = Link Pointer field is not valid.										

9.13.4.2 siTD Endpoint Capabilities / Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

Table 151. Endpoint and Transaction Translator Characteristics

Bit	Description
31	Direction (I/O). 0 = OUT 1 = IN. This field encodes whether the full-speed transaction should be an IN or OUT.
30:24	Port Number. This field is the port number of the recipient Transaction Translator.
23	(Reserved). Bit reserved and should be zero.
22:16	Hub Address. This field holds the device address of the Companion Controllers' hub.
15:12	(Reserved). Field reserved and should be zero.
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	(Reserved). Bit is reserved for future use. It should be zero.
6:0	Device Address. This field selects the specific device serving as the data source or sink.

**Table 152. Micro-Frame Schedule Control**

Bit	Description
31:16	(Reserved). This field reserved for future use. It should be zero.
15:8	Split Completion Mask (μFrame C-Mask). This field (along with the <i>Active</i> and <i>SplitX-state</i> fields in the <i>Status</i> byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame C-Mask field is a one, then this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7:0	Split Start Mask (μFrame S-mask). This field (along with the <i>Active</i> and <i>SplitX-state</i> fields in the <i>Status</i> byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

9.13.4.3 siTD Transfer State

DWords 3-6 are used to manage the state of the transfer.

Table 153. siTD Transfer Status and Control (Sheet 1 of 2)

Bit	Description
31	Interrupt On Complete (ioc). 0 = Do not interrupt when transaction is complete. 1 = Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed it will assert a hardware interrupt at the next interrupt threshold.
30	Page Select (P). Used to indicate which data page pointer should be concatenated with the <i>CurrentOffset</i> field to construct a data buffer pointer (0 selects <i>Page 0</i> pointer and 1 selects <i>Page 1</i>). The host controller is not required to write this field back when the siTD is retired (<i>Active</i> bit transitioned from a one to a zero).
29:26	(Reserved). This field reserved for future use and should be zero.
25:16	Total Bytes To Transfer. This field is initialized by software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)
15:8	μFrame Complete-split Progress Mask (C-prog-Mask). This field is used by the host controller to record which split-completes has been executed.
7:0	Status. This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding
7	Active. Set to one by software to enable the execution of an isochronous split transaction by the Host Controller.
6	ERR. Set to a one by the Host Controller when an ERR response is received from the Companion Controller.
5	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the Host Controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.
4	Babble Detected. Set to a one by the Host Controller during status update when "babble" is detected during the transaction generated by this descriptor.
3	Transaction Error (XactErr). Set to a one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, etc.). This bit will only be set for IN transactions.



Table 153. siTD Transfer Status and Control (Sheet 2 of 2)

Bit	Description
2	Missed Micro-Frame. The host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction.
1	Split Transaction State (SplitXstate). The bit encodings are: Value Meaning 00b Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask. 01b Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask.
0	(Reserved). Bit reserved for future use and should be zero.

9.13.4.4 siTD Buffer Pointer List (Plus)

DWords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most significant 20 bits of each DWord in this section are the 4K (page) aligned buffer pointers. The least significant 12 bits of each DWord are used as additional transfer state.

Table 154. Buffer Page Pointer List (Plus)

Bit	Description
31:12	Buffer Pointer List. Bits [31:12] of DWords 4 and 5 are 4K paged aligned, physical memory addresses. These bits correspond to physical address bits [31:12] respectively. The lower 12 bits in each pointer are defined and used as specified below. The field <i>P</i> specifies the <i>current</i> active pointer
11:0	Page 0: Current Offset. The 12 least significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (<i>P</i> field)). The host controller is not required to write this field back when the siTD is retired (<i>Active</i> bit transitioned from a one to a zero). The least significant bits of Page 1 pointer is split into three sub-fields Page 1:
11:5	(Reserved).
4:3	Transaction position (TP). This field is used with T-count to determine whether to send <i>all</i> , <i>first</i> , <i>middle</i> , or <i>last</i> with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: Value Meaning 00b All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01b Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10b Mid. This is the <i>middle</i> payload for a full-speed OUT transaction that is larger than 188 bytes. 11b End. This is the <i>last</i> payload for a full-speed OUT transaction that was larger than 188 bytes.
2:0	Transaction count (T-Count). Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

9.13.4.5 siTD Back Link Pointer

DWord 6 of a siTD is simply another schedule link pointer. This pointer is always zero, or references a siTD. This pointer cannot reference any other schedule data structure.


Table 155. siTD Back Link Pointer

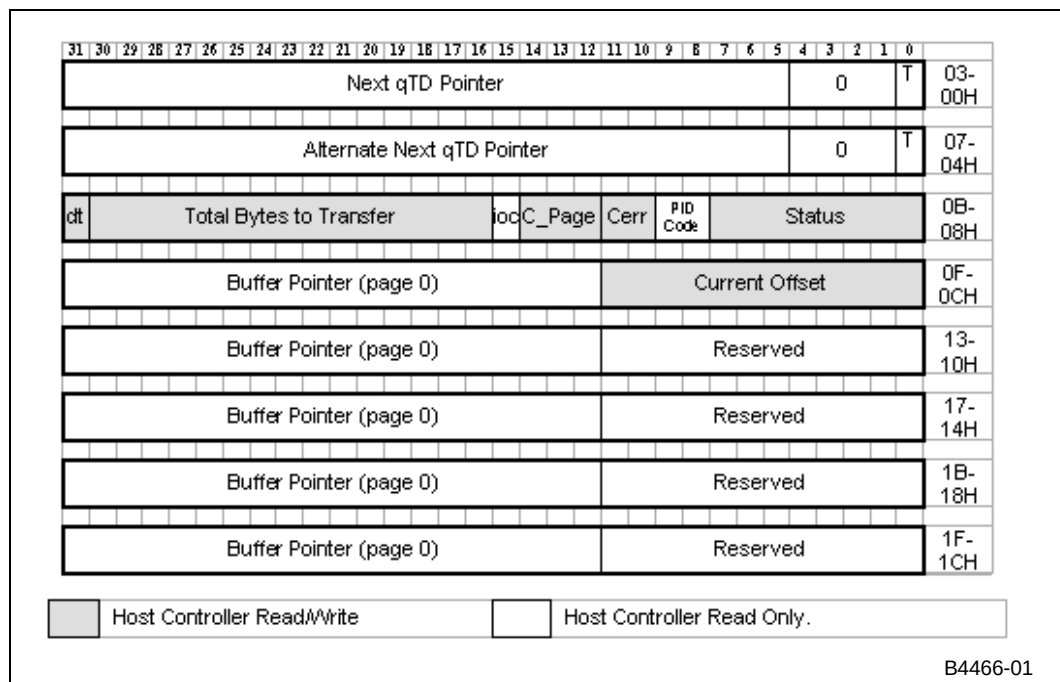
Bit	Description
31:5	siTD Back Pointer. This field is a physical memory pointer to a siTD.
4:1	(Reserved). This field is reserved for future use. It should be zero.
0	Terminate (T). 0 = siTD Back Pointer field is valid. 1 = siTD Back Pointer field is not valid.

9.13.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5*4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

Figure 49. Queue Element Transfer Descriptor Block Diagram


Queue Element Transfer Descriptors must be aligned on 32-byte boundaries.



9.13.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 156. qTD Next Element Transfer Pointer (DWord 0)

Bit	Description
31:5	Next Transfer Element Pointer. This field contains the physical memory address of the next qTD to be processed. The field corresponds to memory address signals[31:5], respectively.
4:1	(Reserved). These bits are reserved and their value has no effect on operation.
0	Terminate (T). 0 = Pointer is valid (points to a valid Transfer Element Descriptor). 1 = pointer is invalid. This bit indicates to the Host Controller that there are no more valid entries in the queue.

9.13.5.2 Alternate Next qTD Pointer

The second DWord of a queue element transfer descriptor is used to support hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit the host controller will always use this pointer when the current qTD is retired due to short packet.

Table 157. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bit	Description
31:5	Alternate Next Transfer Element Pointer. This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively.
4:1	(Reserved). These bits are reserved and their value has no effect on operation.
0	Terminate (T). 0 = Pointer is valid (points to a valid Transfer Element Descriptor). 1 = pointer is invalid. This bit indicates to the Host Controller that there are no more valid entries in the queue.

9.13.5.3 qTD Token

The third DWord of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head).

Note: The field descriptions forward reference fields defined in the queue head. Where necessary, these forward references are preceded with a QH notation.



Table 158. qTD Token (DWord 2) (Sheet 1 of 2)

Bit	Description													
31	Data Toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the <i>Data Toggle Control</i> bit in the queue head.													
30:16	Total Bytes to Transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value software may store in this field is 5 * 4K (5000H). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that <i>Total Bytes To Transfer</i> be an even multiple of QHD.Maximum Packet Length. If software builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than QHD.Maximum Packet Length. Although it is possible to create a transfer up to 20K this assumes the 1 st offset into the first page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K*. Therefore, the maximum recommended transfer is 16K(4000H).													
15	Interrupt On Complete (IOC). If this bit is set to a one, it specifies that when this qTD is completed, the Host Controller should issue an interrupt at the next interrupt threshold.													
14:12	Current Page (C_Page). This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0H to 4H. The host controller is not required to write this field back when the qTD is retired.													
11:10	Error Counter (CERR). This field is a 2-bit down counter that keeps track of the number of consecutive Errors detected while executing this qTD. If this field is programmed with a non-zero value during set-up, the Host Controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the Host Controller marks the qTD inactive, sets the <i>Halted</i> bit to a one, and error status bit for the error that caused <i>CERR</i> to decrement to zero. An interrupt will be generated if the <i>USB Error Interrupt Enable</i> bit in the USBINTR register is set to a one. If HCD programs this field to zero during set-up, the Host Controller will not count errors for this qTD and there will be no limit on the retries of this qTD. Note that write-backs of intermediate execution state are to the queue head overlay area, not the qTD. <table><tr><td>Error</td><td>Decrement Counter</td></tr><tr><td>Transaction Error</td><td>Yes</td></tr><tr><td>Data Buffer Error</td><td>No³</td></tr><tr><td>Stalled</td><td>No¹</td></tr><tr><td>Babble Detected</td><td>No¹</td></tr><tr><td>No Error</td><td>No²</td></tr></table>		Error	Decrement Counter	Transaction Error	Yes	Data Buffer Error	No ³	Stalled	No ¹	Babble Detected	No ¹	No Error	No ²
Error	Decrement Counter													
Transaction Error	Yes													
Data Buffer Error	No ³													
Stalled	No ¹													
Babble Detected	No ¹													
No Error	No ²													
	1	Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented												
	2	If the EPS field indicates a HS device or the queue head is in the Asynchronous Schedule (and <i>PIDCode</i> indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset <i>CERR</i> to extend the total number of errors for this transaction. For example, <i>CERR</i> should be reset with maximum value (3) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 00b. See " Split Transaction Execution State Machine for Interrupt " on page 461 for <i>CERR</i> adjustment rules when the EPS field indicates a FS or LS device and the queue head is in the Periodic Schedule. See " Asynchronous — Do Complete Split " on page 454 for <i>CERR</i> adjustment rules when the EPS field indicates a FS or LS device, the queue head is in the Asynchronous schedule and the <i>PIDCode</i> indicates a SETUP.												
	3	Data buffer errors are host problems. They don't count against the device's retries.												
	Note: Software must not program CERR to a value of zero when the EPS field is programmed with a value indicating a Full- or Low-speed device. This combination could result in undefined behavior.													
9:8	PID Code. This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:													
	00b	OUT Token generates token (E1H)												
	01b	IN Token generates token (69H)												



Table 158. qTD Token (DWord 2) (Sheet 2 of 2)

Bit	Description	
	10b	SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt the queue head is non-zero.) transfer type, e.g. <i>μFrame S-mask</i> field in
	11b	(Reserved)
7:0	Status. This field is used by the Host Controller to communicate individual command execution states back to HCD. This field contains the status of the last transaction performed on this qTD. The bit encodings are:	
	7	Active. Set to one by software to enable the execution of transactions by the Host Controller.
	6	Halted. Set to a one by the Host Controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set to a one, the Active bit is also set to zero.
	5	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the Host Controller will force a timeout condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	4	Babble Detected. Set to a one by the Host Controller during status update when "babble" is detected during the transaction. In addition to setting this bit, the Host Controller also sets the <i>Halted</i> bit to a one. Since "babble" is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.
	3	Transaction Error (XactErr). Set to a one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, etc.). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	2	Missed Micro-Frame. This bit is ignored unless the <i>QH.EPS</i> field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	1	Split Transaction State (SplitXstate). This bit is ignored by the host controller unless the <i>QH.EPS</i> field indicates a full- or low-speed endpoint. When a Full- or Low-speed device, the host controller uses this bit to track the state of the split-transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are: Value Meaning 0b Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint. 1b Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint.
	0	Ping State (P) /ERR. If the <i>QH.EPS</i> field indicates a High-speed device and the <i>PID_Code</i> indicates an OUT endpoint, then this is the state bit for the Ping protocol. The bit encodings are: Value Meaning 0b Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1b Do Ping. This value directs the host controller to issue a PING PID to the endpoint. If the <i>QH.EPS</i> field does not indicate a High-speed device, then this field is used as an error indicator bit. It is set to a one by the host controller whenever a periodic split-transaction receives an ERR handshake.



9.13.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor is an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes *Current Offset* field to the starting offset into the current page, where current page is selected via the value in the *C_Page* field.

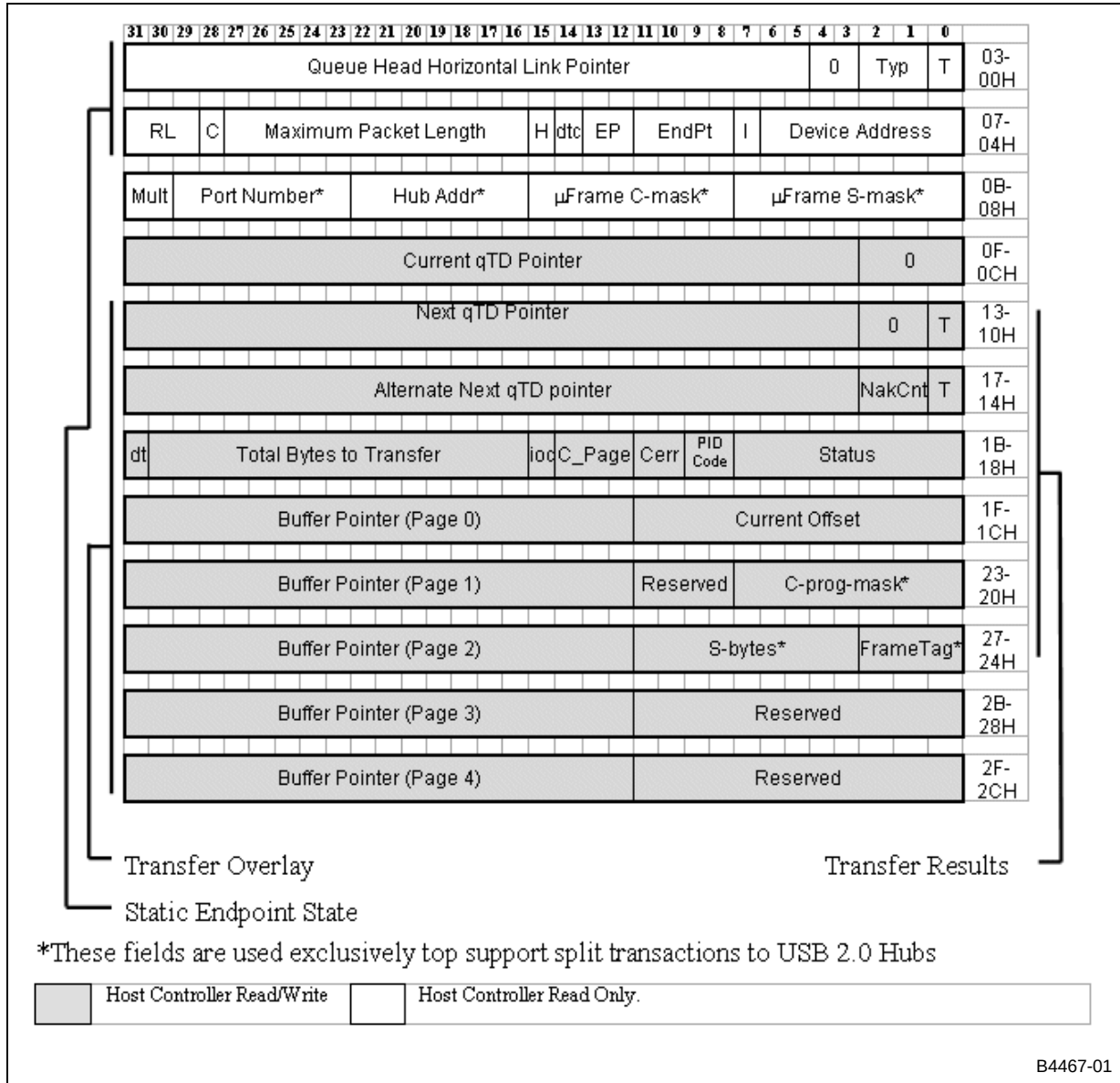
Table 159. qTD Buffer Pointer(s) (DWords 3-7)

Bit	Description
31:12	Buffer Pointer List. Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field <i>C_Page</i> specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using <i>C_Page</i> (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment <i>C_Page</i> and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11:0	Current Offset (Reserved). This field is reserved in all pointers except the first one (e.g. Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by <i>C_Page</i>). The host controller is not required to write this field back when the qTD is retired. Software should ensure the Reserved fields are initialized to zeros.



9.13.6 Queue Head

Figure 50. Queue Head Structure Layout



9.13.6.1 Queue Head Horizontal Link Pointer

The first DWord of a Queue Head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

**Table 160. Queue Head DWord 0**

Bit	Description										
31:5	Queue Head Horizontal Link Pointer (QHLP). This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively.										
4:3	(Reserved). These bits must be written as zeros.										
2:1	QH/(s)ITD Select (Typ). This field indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>00b</td><td>iTD (isochronous transfer descriptor)</td></tr> <tr> <td>01b</td><td>QH (queue head)</td></tr> <tr> <td>10b</td><td>siTD (split transaction isochronous transfer descriptor)</td></tr> <tr> <td>11b</td><td>FSTN (frame span traversal node)</td></tr> </table>	Value	Meaning	00b	iTD (isochronous transfer descriptor)	01b	QH (queue head)	10b	siTD (split transaction isochronous transfer descriptor)	11b	FSTN (frame span traversal node)
Value	Meaning										
00b	iTD (isochronous transfer descriptor)										
01b	QH (queue head)										
10b	siTD (split transaction isochronous transfer descriptor)										
11b	FSTN (frame span traversal node)										
0	Terminate (T). 0 = Pointer is valid. 1 = Last QH (pointer is invalid). If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the Asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.										

9.13.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specifies static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- **Endpoint Characteristics.** These are the USB endpoint characteristics including addressing, maximum packet size, and endpoint speed.
- **Endpoint Capabilities.** These are adjustable parameters of the endpoint. They effect how the endpoint data stream is managed by the host controller.
- **Split Transaction Characteristics.** This data structure is used to manage full- and low-speed data streams for bulk, control, and interrupt via split transactions to USB 2.0 hub Transaction Translator. There are additional fields used for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 161. Endpoint Characteristics: Queue Head DWord 1 (Sheet 1 of 2)

Bit	Description
31:28	Nak Count Reload (RL). This field contains a value, which is used by the host controller to reload the Nak Counter field.
27	Control Endpoint Flag (C). If the <i>QH.EPS</i> field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26:16	Maximum Packet Length. This directly corresponds to the maximum packet size of the associated endpoint (<i>wMaxPacketSize</i>). The maximum value this field may contain is 0x400 (1024).
15	Head of Reclamation List Flag (H). This bit is set by System Software to mark a queue head as being the head of the reclamation list.



Table 161. Endpoint Characteristics: Queue Head DWord 1 (Sheet 2 of 2)

Bit	Description	
14	Data Toggle Control (DTC). This bit specifies where the host controller should get the initial data toggle on an overlay transition. 0b Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1b Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.	
13:12	Endpoint Speed (EPS). This is the speed of the associated endpoint. Bit combinations are:	
	00b	Full-Speed (12Mbs)
	01b	Low-Speed (1.5Mbs)
	10b	High-Speed (480 Mb/s)
	11b	(Reserved)
	This field must not be modified by the host controller.	
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.	
7	Inactivate on Next Transaction (I). This bit is used by system software to request that the host controller set the Active bit to zero. See “Rebalancing the Periodic Schedule” on page 467 for full operational details. This field is only valid when the queue head is in the Periodic Schedule and the EPS field indicates a Full or Low-speed endpoint. Setting this bit to a one when the queue head is in the Asynchronous Schedule or the EPS field indicates a high-speed device yields undefined results.	
6:0	Device Address. This field selects the specific device serving as the data source or sink.	

Table 162. Endpoint Capabilities: Queue Head DWord 2 (Sheet 1 of 2)

Bit	Description	
31:30	High-Bandwidth Pipe Multiplier (Mult). This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). The valid values are:	
	Value	Meaning
	00b	(Reserved). A zero in this field yields undefined results.
	01b	One transaction to be issued for this endpoint per micro-frame
	10b	Two transactions to be issued for this endpoint per micro-frame
11b	Three transactions to be issued for this endpoint per micro-frame	
29:23	Port Number. This field is ignored by the host controller unless the <i>EPS</i> field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address <i>Hub Addr</i> below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.	

**Table 162. Endpoint Capabilities: Queue Head DWord 2 (Sheet 2 of 2)**

Bit	Description
22:16	Hub Addr. This field is ignored by the host controller unless the <i>EPS</i> field indicates a full- or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.
15:8	Split Completion Mask (μFrame C-Mask). This field is ignored by the host controller unless the <i>EPS</i> field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the <i>Active</i> and <i>SplitX-state</i> fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the <i>FRINDEX</i> register. If the <i>FRINDEX</i> register bits decode to a position where the μFrame C- Mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7:0	Interrupt Schedule Mask (μFrame S-mask). This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the <i>FRINDEX</i> register as an index into a bit position in this bit vector. If the μFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the <i>EPS</i> field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the <i>PID_Code</i> field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the <i>EPS</i> field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

9.13.6.3 Transfer Overlay

The nine DWords in this area represent a *transaction working space* for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the *Queue Head Horizontal Link Pointer* to the next queue head. The host controller will never follow the *Next Transfer Queue Element* or *Alternate Queue Element* pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The DWord3 of a Queue Head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 163. Current qTD Link Pointer

Bit	Description
31:5	Current Element Transaction Descriptor Link Pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively.
4:0	(Reserved) (R). These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The DWords 4-11 of a queue head are the transaction overlay area. This area has the same base structure as a Queue Element Transfer Descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an *overlay* because when the queue is advanced to the next queue element, the source queue element is *merged* onto this area. This area serves an execution cache for the transfer.



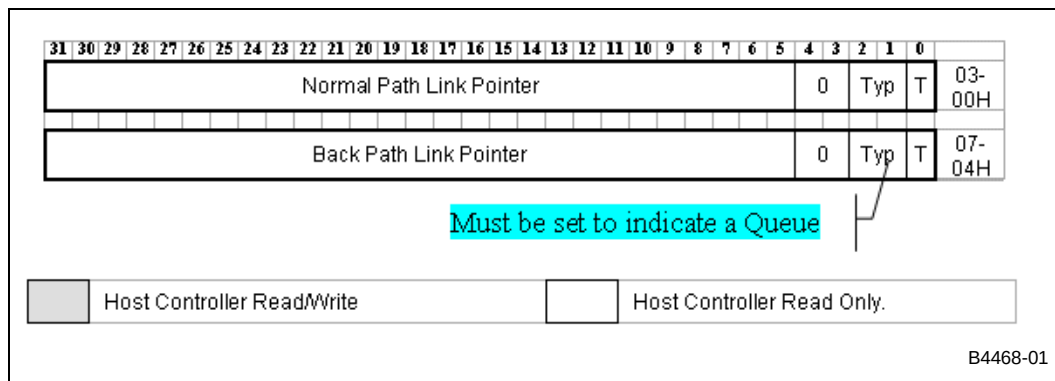
Table 164. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)

DWord	Bit	Description
5	4:1	Nak Counter (NakCnt) _{uRW} . This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay.
6	31	Data Toggle . The <i>Data Toggle Control</i> controls whether the host controller preserves this bit when an overlay operation is performed.
6	15	Interrupt On Complete (IOC) . The IOC control bit is always inherited from the source qTD when the overlay operation is performed.
6	11:10	Error Counter (C_ERR) . This two-bit field is copied from the qTD during the overlay and written back during queue advancement.
6	0	Ping State (P) /ERR . If the <i>EPS</i> field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	7:0	Split-transaction Complete-split Progress (C-prog-mask) . This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	4:0	Split-transaction Frame Tag (Frame Tag) . This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	11:5	S-bytes . Software must ensure that the <i>S-bytes</i> field in a qTD is zero before activating the qTD. This field is used to keep track of the number of bytes sent or received during an IN or OUT split transaction.

9.13.7 Periodic Frame Span Traversal Node (FSTN)

This data structure is to be used only for managing Full- and Low-speed transactions that span a Host-frame boundary. See “[Host Controller Operational Model for FSTNs](#)” on page 458 for full operational details. Software must not use an FSTN in the Asynchronous Schedule. An FSTN in the Asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation below 0096h. FSTNs are not defined for implementations before 0.96 and their use will yield undefined results.

Figure 51. Frame Span Traversal Node Structure Layout



9.13.7.1 FSTN Normal Path Pointer

The first DWord of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

**Table 165. FSTN Normal Path Pointer Signals**

Bit	Description
31:5	Normal Path Link Pointer (NPLP). This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as 0s.
2:1	QH / (s)iTD / FSTN Select (Typ). This field indicates to the Host Controller whether the item referenced is a iTD/siTD, a QH or an FSTN. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor) 11b FSTN (Frame Span Traversal Node)
0	Terminate (T). 0 = Link Pointer is valid. 1 = Link Pointer field is not valid.

9.13.7.2 FSTN Back Path Link Pointer

The second DWord of an FTSN node contains a link pointer to a queue head. If the *T-bit* in this pointer is a zero, then this FSTN is a *Save-Place* indicator. Its *Typ* field must be set by software to indicate the target data structure is a queue head. If the *T-bit* in this pointer is set to a one, then this FSTN is the *Restore* indicator. When the *T-bit* is a one, the host controller ignores the *Typ* field.

Table 166. FSTN Back Path Link Pointer Signals

Bit	Description
31:5	Back Path Link Pointer (BPLP). This field contains the address of a Queue Head. This field corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as 0s.
2:1	Typ. Software must ensure this field is set to indicate the target data structure is a Queue Head. Any other value in this field yields undefined results.
0	Terminate (T). 0 = Link Pointer is valid (i.e. the host controller may use bits [31:5] (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a <i>Save-Place</i> indicator. 1 = Link Pointer field is not valid (i.e. the host controller must not use bits [31:5] (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a <i>Restore</i> indicator.

9.14 Host Operational Model

The general operational model is for the enhanced interface host controller hardware and enhanced interface host controller driver (generally referred to as system software). Each significant operational feature of the EHCI host controller is discussed in a separate section. Each section presents the operational model requirements for the host controller hardware. Where appropriate, recommended system software operational models for features are also presented.



9.14.1 Host Controller Initialization

After initial power-on or *HCReset* (hardware or via *HCReset* bit in the *USBCMD* register), all of the operational registers will be at their default values, as illustrated in Table 167, “Default Values of Operational Register Space” on page 410. After a hardware reset, only the operational registers not contained in the Auxiliary power well will be at their default values.

Table 167. Default Values of Operational Register Space

Operational Register	Default Value (after Reset)
USBCMD	0x00080000 (0x00080B00 if <i>Asynchronous Schedule Park Capability</i> is a one)
USBSTS	0x00001000
USBINTR	0x00000000
FRINDEX	0x00000000
CTRLDSSEGMENT	0x00000000
PERIODICLISTBASE	Undefined
ASYNCLISTADDR	Undefined
CONFIGFLAG	0x00000000
PORTSC	0x00002000 (w/PPC set to one); 0x00003000 (w/PPC set to a zero)

In order to initialize the host controller, software should perform the following steps:

- Program the *CTRLDSSEGMENT* register with 4-Gigabyte segment where all of the interface data structures are allocated.
- Write the appropriate value to the *USBINTR* register to enable the appropriate interrupts.
- Write the base address of the Periodic Frame List to the *PERIODICLIST BASE* register. If there are no work items in the periodic schedule, all elements of the Periodic Frame List should have their *T-Bits* set to a one.
- Write the *USBCMD* register to set the desired interrupt threshold, frame list size (if applicable) and turn the host controller *ON* via setting the *Run/Stop* bit.
- Write a 1 to *CONFIGFLAG* register to route all ports to the EHCI controller (see Section 9.14.2, “Port Routing and Control” on page 411).

At this point, the host controller is up and running and the port registers will begin reporting device connects, etc. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled High-speed ports, but the schedules have not yet been enabled. The EHCI Host controller will not transmit SOFs to enabled Full- or Low-speed ports. In order to communicate with devices via the asynchronous schedule, system software must write the *ASYNCLISTADDR* register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to the *Asynchronous Schedule Enable* bit in the *USBCMD* register. In order to communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to the *Periodic Schedule Enable* bit in the *USBCMD* register. Note that the schedules can be turned on before the first port is reset (and enabled).

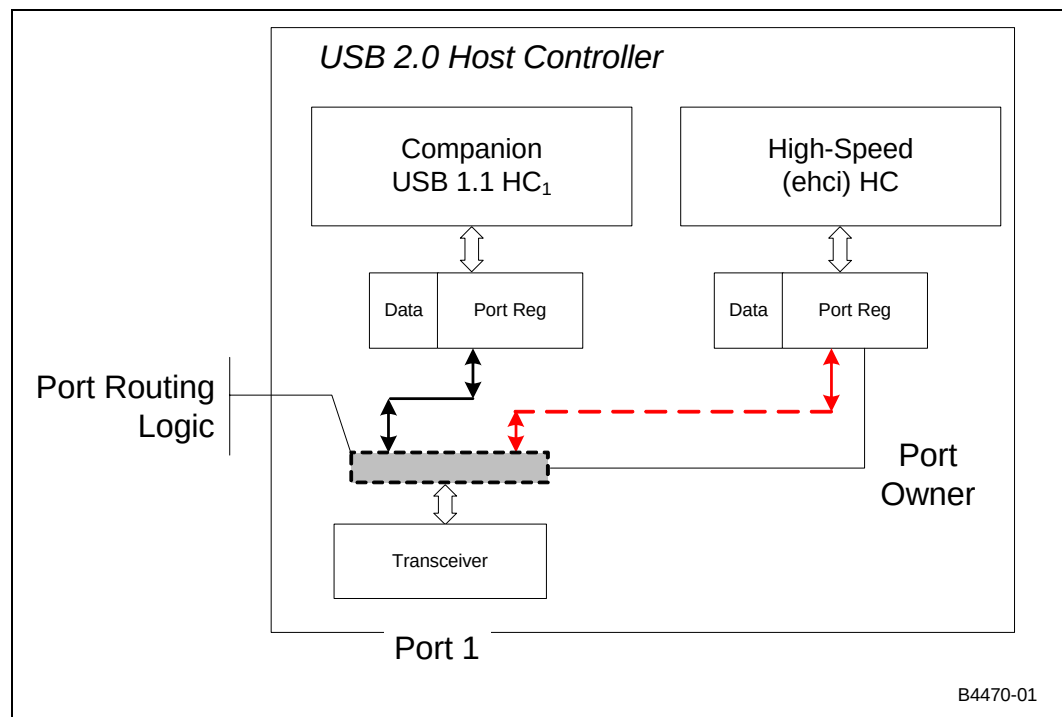
Any time the *USBCMD* register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.



9.14.2 Port Routing and Control

A USB 2.0 Host controller is comprised of one high-speed host controller, which implements the EHCI programming interface and 0 to N USB 1.1 companion host controllers. Companion host controllers (chCs) may be implementations of either Universal or Open host controller specifications. This configuration is used to deliver the required full USB 2.0-defined port capability; e.g. Low-, Full-, and High-speed capability for every port. “[Example USB 2.0 Host Controller Port Routing Block Diagram](#)” on [page 411](#) illustrates a simple block diagram of the port routing logic and its relationship to the high-speed and companion host controllers within a USB 2.0 Host controller.

Figure 52. Example USB 2.0 Host Controller Port Routing Block Diagram



There is one transceiver per physical port and each host controller module has its own port status and control registers. The EHCI controller has port status and control registers for every port. Each companion host controller has only the port control and status registers it is required to operate. Each transceiver can be controlled by either the EHCI host controller or one companion host controller. Routing logic lies between the transceiver and the port status and control registers.

The port routing logic is controlled from signals originating in the EHCI host controller. The EHCI host controller has a *global* routing policy control field and per-port *ownership* control fields. The *Configured Flag (CF)* bit (defined in [Section 9.12.8, “BURSTSIZE” on page 380](#)) is the global routing policy control. At power-on or reset, the default routing policy is to the companion controllers (if they exist). If the system does not include a driver for the EHCI host controller and the host controller includes Companion Controllers, then the ports will still work in Full- and Low-speed mode (assuming the system includes a driver for the companion controllers). In general, when the EHCI owns the ports, the companion host controllers' port registers do not see a connect indication from the transceiver. Similarly, when a companion host controller owns a port, the EHCI controller's port registers do not see a connect indication from the transceiver. The details on the rules for the port routing logic are described in the



following sections. The USB 2.0 host controller must be implemented as a multi-function PCI device if the implementation includes companion controllers. The companion host controllers' function numbers must be less than the EHCI host controller function number. The EHCI host controller must be a larger function number with respect to the companion host controllers associated with this EHCI host controller. If a PCI device implementation contains only an EHCI controller (i.e. no companion controllers or other PCI functions), then the EHCI host controller must be function zero, in accordance with the PCI Specification. The *N_CC* field in the Structural Parameter register (*HCSPARAMS*) indicates whether the controller implementation includes companion host controllers. When *N_CC* has a non-zero value there exists companion host controllers. If *N_CC* has a value of zero, then the host controller implementation does not include companion host controllers. If the host controller root ports are exposed to attachment of full- or low-speed devices, the ports will always fail the high-speed chirp during reset and the ports will not be enabled. System software can notify the user of the illegal condition. This type of implementation requires a USB 2.0 hub be connected to a root port to provide full and low-speed device connectivity.

System software uses information in the host controller capability registers to determine how the ports are routed to the companion host controllers. See [Section 9.11.3, "HCSPARAMS – EHCI Compliant with Extensions" on page 370](#).

9.14.2.1 Port Routing Control via EHCI Configured (CF) Bit

Each port in the USB 2.0 host controller can be routed either to a single companion host controller or to the EHCI host controller. The port routing logic is controlled by two mechanisms in the EHCI HC: a host controller global flag and per-port control. The *Configured Flag (CF)* bit (defined in [Section 9.12.8, "BURSTSIZE" on page 380](#)), is used to globally set the policy of the routing logic. Each port register has a *Port Owner* control bit which allows the EHCI Driver to explicitly control the routing of individual ports. Whenever the *CF bit* transitions from a zero to a one (this transition is only available under program control) the port routing unconditionally routes all of the port registers to the EHCI HC (all *Port Owner* bits go to zero). While the *CF-bit* is a one, the EHCI Driver can control individual ports' routing via the *Port Owner* control bit. Likewise, whenever the *CF bit* transitions from a one to a zero (as a result of Aux power application, *HCRESET*, or software writing a zero to *CF-bit*), the port routing unconditionally routes all of the port registers to the appropriate companion HC. The default value for the EHCI HC's *CF bit* (after Aux power application or *HCRESET*) is zero. [Table 168, "Default Port Routing Depending on EHCI HC CF Bit" on page 412](#) summarizes the default routing for all the ports, based on the value of the EHCI HC's *CF bit*.

The view of the port depends on the current owner. A Universal or Open companion host controller will see port register bits consistent with the appropriate specification. Port bit definitions that are required for EHCI host controllers are not visible to companion host controllers.

Table 168. Default Port Routing Depending on EHCI HC CF Bit

HS CF Bit	Default Port Ownership	Explanation
0B	Companion HCs	The companion host controllers own the ports and only Full- and Low-speed devices are supported in the system. The ports behave only as Full- and Low-speed ports in this configuration
1B	EHCI HC	The EHCI host controller has default ownership overall of the ports. The routing logic inhibits device connect events from reaching the companion HCs' port status and control registers when the port owner is the EHCI HC. The EHCI HC has access to the additional port status and control bits defined in this specification. The EHCI HC can temporarily release control of the port to a companion HC by setting the <i>PortOwner</i> bit in the <i>PORTSC</i> register to a one.



9.14.2.2 Port Routing Control via *PortOwner* and Disconnect Event

Manipulating the port routing via the *CF-bit* is an extreme process and not intended to be used during normal operation. The normal mode of port ownership transferal is on the granularity of individual ports using the *Port Owner* bit in the EHCI HC's PORTSC register (for hand-offs from EHCI to companion host controllers). Individual port ownership is returned to the EHCI controller when the port registers a device disconnect. When the disconnect is detected, the port routing logic immediately returns the port ownership to the EHCI controller. The companion host controller port register detects the device disconnect and operates normally.

Under normal operating conditions (assuming all HC drivers loaded and operational and the EHCI *CF-bit* is set to a one), the typical port enumeration sequence proceeds as illustrated below:

- Initial condition is that EHCI is port owner. A device is connected causing the port to detect a connect, set the port connect change bit and issue a port-change interrupt (if enabled).
- EHCI Driver identifies the port with the new connect change bit asserted and sends a change report to the hub driver. Hub driver issues a `GetPortStatus()` request and identifies the connect change. It then issues a request to clear the connect change, followed by a request to reset and enable the port.
- When the EHCI Driver receives the request to reset and enable the port, it first checks the value reported by the *LineStatus* bits in the PORTSC register. If they indicate the attached device is a full-speed device (e.g. D+ is asserted), then the EHCI Driver sets the *PortReset* control bit to a one (and sets the *PortEnable* bit to a zero) which begins the reset-process. Software times the duration of the reset, then terminates reset signaling by writing a zero to the port reset bit. The reset process is actually complete when software reads a zero in the *PortReset* bit. The EHCI Driver checks the *PortEnable* bit in the PORTSC register. If set to a one, the connected device is a high-speed device and EHCI Driver (root hub emulator) issues a change report to the hub driver and the hub driver continues to enumerate the attached device.
- At the time the EHCI Driver receives the port reset and enable request the *LineStatus* bits might indicate a low-speed device. Additionally, when the port reset process is complete, the *PortEnable* field may indicate that a full-speed device is attached. In either case the EHCI driver sets the *PortOwner* bit in the PORTSC register to a one to release port ownership to a companion host controller.
- When the EHCI Driver sets *PortOwner* bit to a one, the port routing logic makes the connection state of the transceiver available to the companion host controller port register and removes the connection state from the EHCI HC port. The EHCI PORTSC register observes and reports a disconnect event via the disconnect change bit. The EHCI Driver detects the connection status change (either by polling or by port change interrupt) and then sends a change report to the hub driver. When the hub driver requests that port-state, the EHCI Driver responds with a reset complete change set to a one, a connect change set to a one and a connect status set to a zero. This information is derived directly from the EHCI port register. This will allow the hub driver to assume the device was disconnected during reset. It will acknowledge the change bits and wait for the next change event. While the EHCI controller does not own the port, it simply remains in a state where the port reports no device connected.

The device-connect evaluation circuitry of the companion HC activates and detects the device, the companion Driver detects the connection and enumerates the port.

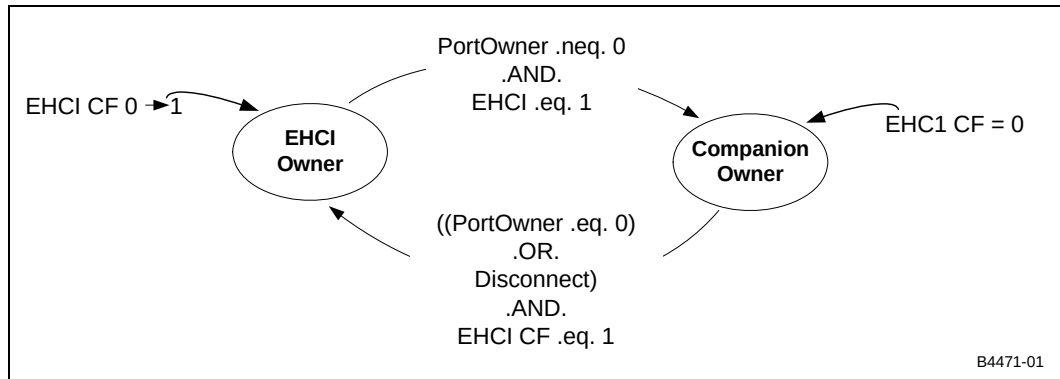
When a port is routed to a companion HC, it remains under the control of the companion HC until the device is disconnected from the root port (ignoring for now the scenario where EHCI's *CF-bit* transitions from a 1b to a 0b). When a disconnect occurs, the disconnect event is detected by both the companion HC port control and the EHCI port ownership control. On the event, the port ownership is returned immediately to

the EHCI controller. The companion HC stack detects the disconnect and acknowledges as it would in an ordinary standalone implementation. Subsequent connects will be detected by the EHCI port register and the process will repeat.

9.14.2.3 Example Port Routing State Machine

Figure 53, “Port Owner Hand-Off State Machine” on page 414 illustrates an example of how the port ownership should be managed. The following sections describe the entry conditions to each state.

Figure 53. Port Owner Hand-Off State Machine



9.14.2.3.1 EHCI HC Owner

Entry to this state occurs whenever one of the following events occur:

- When the EHCI HC's *Configure Flag (CF)* bit in the CONFIGFLAG register transitions from a zero to a one. This signals the fact that the system has a host controller driver for the EHCI HC and that all ports in the USB 2.0 host controller must default route to the EHCI controller.
- When the port is owned by a companion HC and the device is disconnected from the port. The EHCI port routing control logic is notified of the disconnect, and returns port routing to the EHCI controller. The connection state of the companion HC goes immediately to the disconnected state (with appropriate side effect to connect change, enable and enable change). The companion HC driver will acknowledge the disconnect by setting the connect status change bit to a zero. This allows the companion HC's driver to interact with the port completely through the disconnect process.
- When system software writes a zero to the *PortOwner* bit in the PORTSC register. This allows software to take ownership of a port from a companion host controller. When this occurs, the routing logic to the companion HC effectively signals a disconnect to the companion HC's port status and control register.

9.14.2.3.2 Companion HC Owner

Entry to this state occurs whenever one of the following events occur:

- When the *Port Owner* field transitions from a zero to a one.
- When the HS-mode HC's *Configure Flag (CF)* is equal to zero.

On entry to this state, the routing logic allows the companion HC port register to detect a device connect. Normal port enumeration proceeds.



9.14.2.4 Port Power

The *Port Power Control (PPC)* bit in the HCSPARAMS register indicates whether the USB 2.0 host controller has port power control (See [Section 9.11.3, “HCSPARAMS – EHCI Compliant with Extensions” on page 370](#)). When this bit is a zero, then the host controller does not support software control of port power switches. When in this configuration, the port power is always available and the companion host controllers must implement functionality consistent with port power always on. When the *PPC* bit is a one, then the host controller implementation includes port power switches. Each available switch has an output enable, which is referred to in this discussion as *PortPowerOutputEnable (PPE)*. *PPE* is controlled based on the state of the combination bits *PPC* bit, *EHCI Configured (CF)*-bit and individual *Port Power (PP)* bits. [Table 169, “Port Power Enable Control Rules” on page 415](#) illustrates the summary behavioral model.

Table 169. Port Power Enable Control Rules

CF	CHC ² (PP)	EHC ³ (PP)	Owner	PPE ¹	Description
0	0	X	CHC	0	When the EHCI controller has not been configured, the port is owned by the companion host controller. When the companion HC's port power select is off, then the port power is off.
0	1	X	CHC	1	Similar to previous entry. When the companion HC's port power select is on, then the port power is on.
1	0	0	CHC	0	Port owner has port power turned off, the power to port is off.
1	0	0	EHC	0	Port owner has port power turned off, the power to port is off.
1	0	1	EHC	1	Port owner has port power on, so power to port is on.
1	0	1	CHC	1	If either HC has port power turned on, the power to the port is on.
1	1	0	EHC	1	If either HC has port power turned on, the power to the port is on.
1	1	0	CHC	1	Port owner has port power on, so power to port is on.
1	1	1	CHC	1	Port owner has port power on, so power to port is on.
1	1	1	EHC	1	Port owner has port power on, so power to port is on.

Notes:

1. PPE (Port Power Enable). This bit actually turns on the port power switch (if one exists).
2. CHC (Companion Host Controller).
3. EHC (EHCI Host Controller).

9.14.2.5 Port Reporting Over-Current

Host controllers are by definition power providers on USB. Whether the ports are considered high- or low-powered is a platform implementation issue. Each EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

The over current detection and limiting logic usually resides outside the host controller logic. This logic may be associated with one or more ports. When this logic detects an over-current condition it is made available to both the companion and EHCI ports. The effect of an over-current status on a companion host controller port is beyond the scope of this document. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- *Over-current Active* bits are set to a one. When the over-current condition goes away, the *Over-current Active* bit will transition from a one to a zero.



- *Over-current Change* bits are set to a one. On every transition of the *Over-current Active* bit the host controller will set the *Over-current Change* bit to a one. Software sets the *Over-current Change* bit to a zero by writing a one to this bit.
- *Port Enabled/Disabled* bit is set to a zero. When this change bit gets set to a one, then the *Port Change Detect* bit in the USBSTS register is set to a one.
- *Port Power (PP)* bits may optionally be set to a zero. There is no requirement in USB that a power provider shut off power in an over current condition. It is sufficient to *limit* the current and leave power applied. When the *Over-current Change* bit transitions from a zero to a one, the host controller also sets the *Port Change Detect* bit in the USBSTS register to a one. In addition, if the *Port Change Interrupt Enable* bit in the USBINTR register is a one, then the host controller will issue an interrupt to the system. Refer to [Table 170, “Behavior During Wake-Up Events” on page 418](#) for summary behavior for over-current detection when the host controller is halted (suspended from a device component point of view).

9.14.3 Suspend / Resume

The EHCI host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely via software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software initiated resumes are called Resume Events/Actions. Bus-initiated resume events are called wake-up events. The classes of wake-up events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, EHCI controllers must always respond to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off by using the per-port control bits in the PORTSC registers.

Selective suspend is a feature supported by every PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the entire bus, it should selectively suspend all enabled ports, then shut off the host controller by setting the *Run/Stop* bit in the USBCMD register to a zero. The EHCI module can then be placed into a lower device state.

When a wake event occurs the system will resume operation and system software will eventually set the *Run/Stop* bit to a one and resume the suspended ports. Software must not set the *Run/Stop* bit to a one until it is confirmed that the clock to the host controller is stable. This is usually confirmed in a system implementation in that all of the clocks in the system are stable before the CPU is restarted. So, by definition, if software is running, clocks in the system are stable and the *Run/Stop* bit in the USBCMD register can be set to a one.

9.14.3.1 Port Suspend / Resume

System software places individual ports into suspend mode by writing a one into the appropriate PORTSC *Suspend* bit. Software must only set the *Suspend* bit when the port is in the enabled state (*Port Enabled* bit is a one) and the EHCI is the port owner (*Port Owner* bit is a zero).

The host controller may evaluate the *Suspend* bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be



several micro-frames of activity on the port until the host controller evaluates the *Suspend* bit. The host controller must evaluate the *Suspend* bit at least every frame boundary.

System software can initiate a resume on a selectively suspended port by writing a one to the *Force Port Resume* bit. Software should not attempt to resume a port unless the port reports that it is in the suspended state (see [Section 9.12.11, "PORTSCx" on page 382](#)). If system software sets *Force Port Resume* bit to a one when the port is not in the suspended state, the resulting behavior is undefined. In order to assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates that it is suspended (*Suspend* bit is a one) before initiating a port resume via the *Force Port Resume* bit. When *Force Port Resume* bit is a one, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then sets the *Force Port Resume* bit to a zero. When the host controller receives the write to transition *Force Port Resume* to zero, it completes the resume sequence as defined in the USB specification, and sets both the *Force Port Resume* and *Suspend* bits to zero. Software-initiated port resumes do not affect the *Port Change Detect* bit in the USBSTS register nor do they cause an interrupt if the *Port Change Interrupt Enable* bit in the USBINTR register is a one. An external USB event may also initiate a resume. The wake events are defined above. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 μ sec. The port's *Force Port Resume* bit is set to a one and the *Port Change Detect* bit in the USBSTS register is set to a one. If the *Port Change Interrupt Enable* bit in the USBINTR register is a one the host controller will issue a hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 ms), then terminates the resume sequence by writing zero to the *Force Port Resume* bit in the port. The host controller receives the write of zero to *Force Port Resume*, terminates the resume sequence and sets *Force Port Resume* and *Suspend* port bits to zero. Software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the *Suspend* and *Force Port Resume* bits are zero. Software must ensure that the host controller is running (i.e. *HCHalted* bit in the USBSTS register is a zero), before terminating a resume by writing a zero to a port's *Force Port Resume* bit. If *HCHalted* is a one when *Force Port Resume* is set to a zero, then SOFs will not occur down the enabled port and the device will return to suspend mode in a maximum of 10 milliseconds.

[Table 170, "Behavior During Wake-Up Events" on page 418](#) summarizes the wake-up events. Whenever a resume event is detected, the *Port Change Detect* bit in the USBSTS register is set to a one. If the *Port Change Interrupt Enable* bit is a one in the USBINTR register, the host controller will also generate an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the *Port Change Detect* status bit in the USBSTS register.

**Table 170. Behavior During Wake-Up Events**

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	Not D0
Port disabled, resume K-State received	No Effect	N/A	N/A
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. Force Port Resume status bit in PORTSC register is set to a one. Port Change Detect bit in USBSTS register set to a one.	1, 2	2
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is a one. A disconnect is detected.	Depending in the initial port state, the PORTSC Connect and Enable status bits are set to zero, and the Connect Change status bit is set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 2	2
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is a zero. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect and Enable status bits are set to zero, and the Connect Change status bit is set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3
Port is not connected and the port's WKCNTNT_E bit is a one. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 2	2
Port is not connected and the port's WKCNTNT_E bit is a zero. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set to a one. If Port Enable/Disable bit is a one, it is set to a zero. Port Change Detect bit in the USBSTS register is set to a one.	1, 2	2
Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set to a one. If Port Enable/Disable bit is a one, it is set to a zero. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3
Notes: 1. Hardware interrupt issued if Port Change Interrupt Enable bit in the USBINTR register is a one. 2. PME# asserted if enabled (Note: PME Status must always be set to a one). 3. PME# not asserted.			

9.14.4 Schedule Traversal Rules

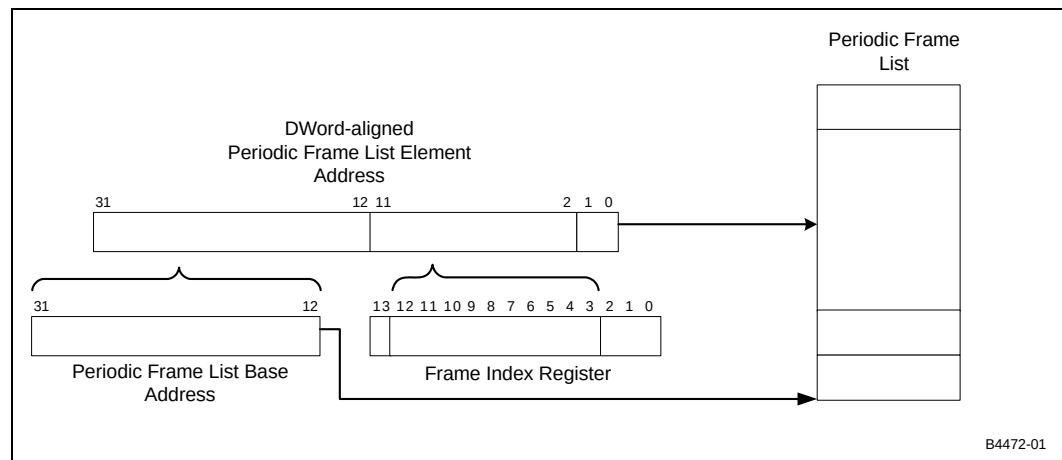
The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by USB, minimize memory traffic and hardware / software complexity.

System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the *PERIODICLISTBASE* register (see [Section 9.12.6, “PERIODICLISTBASE” on page 379](#)). The *PERIODICLISTBASE* register is the physical memory base address of the periodic frame list. The periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in [Section 9.6.2, “Host Data Structure” on page 355](#). In each micro-frame, if the periodic schedule is enabled (see [Section 9.14.6, “Periodic Schedule” on page 425](#)) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the *PERIODICLISTBASE* and the *FRINDEX* registers (see [Figure 54 on page 419](#)). It fetches the element and begins traversing the graph of linked schedule data structures.



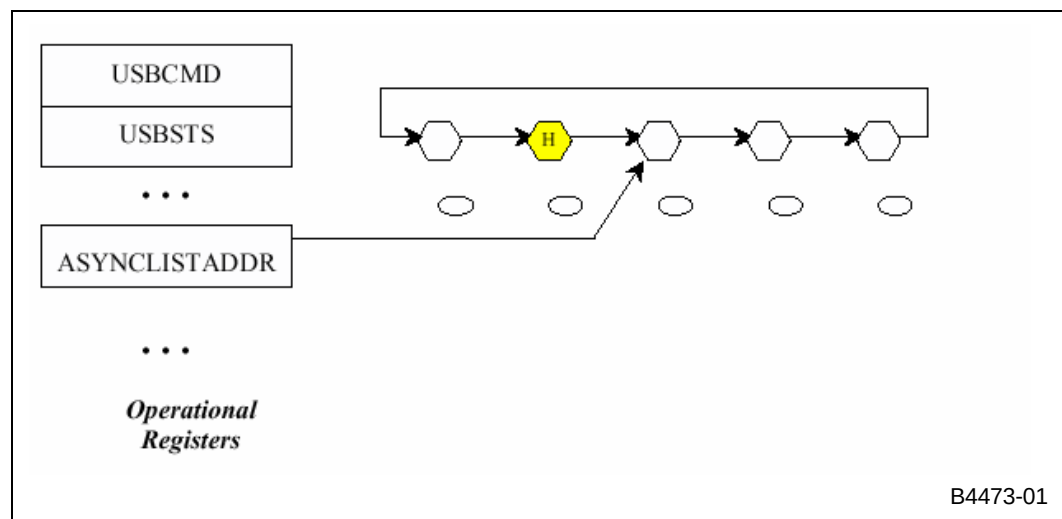
The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set to a one. When the host controller encounters a T-Bit set to a one during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

Figure 54. Derivation of Pointer into Frame List Array



When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, see Figure 55, “General Format of Asynchronous Schedule List” on page 419.

Figure 55. General Format of Asynchronous Schedule List



The ASYNCLISTADDR register contains a physical memory pointer to the *next* queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head *horizontal* pointer T-bits to a zero for queue heads in the asynchronous schedule. See Section 9.14.8, “Asynchronous Schedule” on page 430 for complete operational details.



9.14.4.1 Example: Preserving Micro-Frame Integrity

One of the requirements of a USB host controller is to maintain *Frame Integrity*. This means that the HC must preserve the micro-frame boundaries. For example: SOF packets must be generated on time (within the specified allowable jitter), and High-speed EOF1,2 thresholds must be enforced. The end of micro-frame timing points EOF1 and EOF2 are clearly defined in the USB Specification Revision 2.0. One implication of this responsibility is that the HC must ensure that it does not start transactions that will not be completed before the end of the micro-frame. More precisely, no transactions should be started by the host controller, which cannot be completed in their entirety before the EOF1 point. In order to enforce this rule, the host controller must check each transaction before it starts to ensure that it will complete before the end of the micro-frame.

So, what exactly needs to be involved in this check? Fundamentally, the transaction data payload, plus bit stuffing, plus transaction overhead must be taken into consideration. It is possible to be extremely accurate on how much time the next transaction will take. Take OUTs for an example. The host controller must fetch all of the OUT data from memory in order to send it onto the USB bus. A host controller implementation could pre-fetch *all* of the OUT data, and pre-compute the actual number of bits in the token and data packets. In addition, the system knows the depth of the target endpoint, so it could closely estimate turnaround time for handshake. In addition, the host controller knows the size of a handshake packet. Pre-computing effects of bit stuffing and summing up the other overhead numbers could allow the host controller to know exactly whether there was enough bus time, before EOF1 to complete the OUT transaction. To accomplish this particular approach takes an inordinate amount of time and hardware complexity.

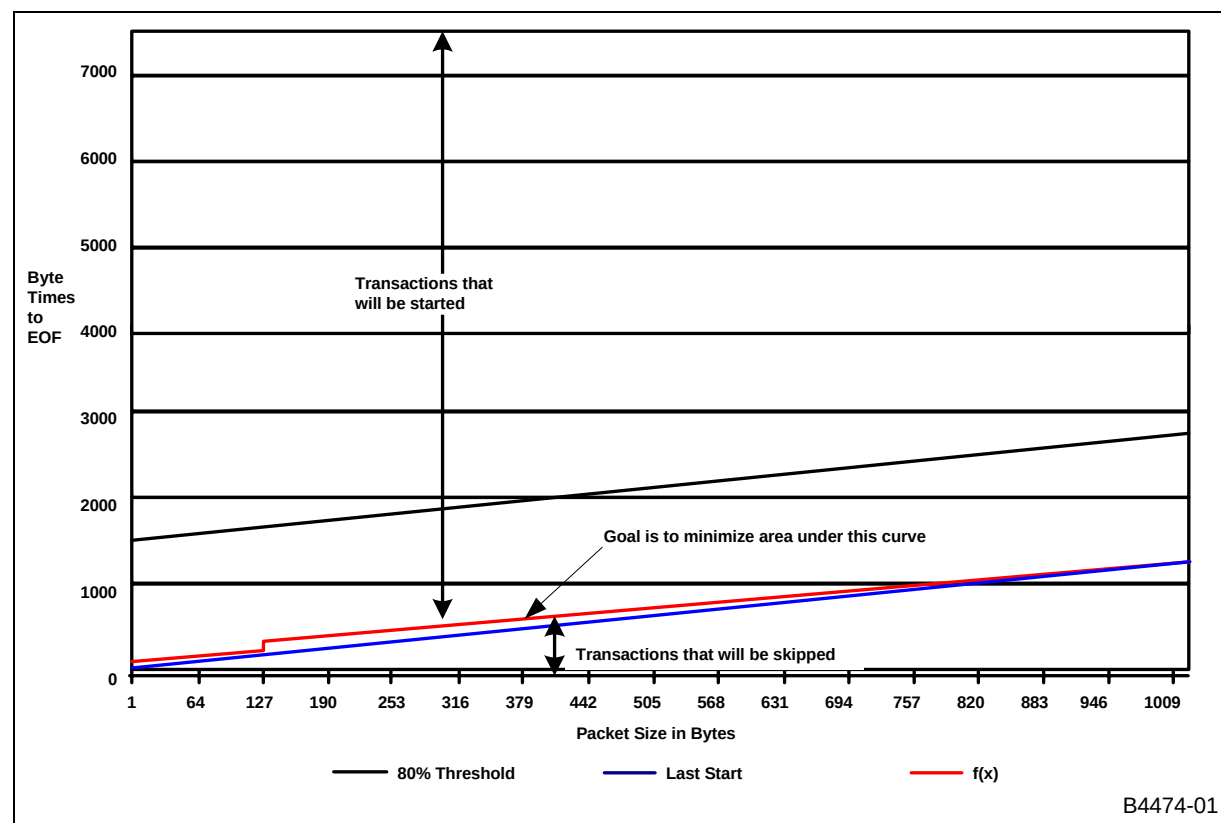
The alternative is to make a reasonable guess whether the next transaction can be started. An example approximation algorithm is described below. This example algorithm relies on the EHCI policy that periodic transactions are scheduled first in the micro-frame. It is a reasonable assumption that software will never over-commit the micro-frame to periodic transactions greater than the specification allowable 80%. In the available remaining 20% bandwidth, the host controller has some ability (in this example) to decide whether or not to execute a transaction. The result of this algorithm is that sometimes, under some circumstances a transaction will not be executed that *could* have been executed. However, under all circumstances, a transaction will never be started unless there is enough time in the frame to complete the transaction.

9.14.4.1.1 Transaction Fit: A Best-Fit Approximation Algorithm

A curve is calculated which represents the *latest* start time for every packet size, at which software will schedule the start of a periodic transaction. This curve is the 80% bandwidth curve. Another curve is calculated which is the absolute, latest permitted start time for every packet size. This curve represents the absolute latest time, that a transaction of each packet size can be started and completed, in the micro-frame. A plot of these two curves is illustrated in [Figure 56, "Best Fit Approximation" on page 421](#). The plot Y-axis represents the number of byte-times left in a frame.

The space between the 80% and the *Last Start* plots is bandwidth reclamation area. In this algorithm the host controller may skip transactions during this time if it is prudent.

The Best-Fit Approximation method plots a function ($f(x)$) between the 80% and *Last Start* curves. The function $f(x)$ adds a constant to every transaction's maximum packet size and the result compared with the number of bytes left in the frame. The constant represents an approximation of the effects of bit stuffing and protocol overhead. The host controller starts transactions whose results land *above* the function curve. The host controller will not start transactions whose results land below the function curve.


Figure 56. Best Fit Approximation


The *LastStart* line was calculated in this example to assume the absolute worst-case bus overhead per transaction. The particular transaction used was a start-split, zero-length OUT transaction with a handshake. A summary of the component parts are listed in [Table 171, “Example Worst-Case Transaction Timing Components” on page 421](#). The component times were derived from the protocol timings defined in the USB Specification Revision 2.0.

Table 171. Example Worst-Case Transaction Timing Components (Sheet 1 of 2)

Component	Bit Time	Byte Time	Explanation
Split Token	76	9.5	Split token as defined in USB core specification. Includes sync, token, eop, etc.
Host 2 Host IPG	88	11	Number of bit times required between consecutive host packets.
Token	67	8.375	Token as defined in USB core specification. Includes sync, token, eop, etc.
Host 2 Host IPG	88	11	Same as above.
Data Packet (0 data bytes)	66.7	8.34	Zero-length data packet. Includes sync, PID, crc16, eop, etc.

**Table 171. Example Worst-Case Transaction Timing Components (Sheet 2 of 2)**

Component	Bit Time	Byte Time	Explanation
Turnaround time	721	90.125	Time for packet initiator (Host) to see the beginning of a response to a transmitted packet.
Handshake packet	48	6	Handshake packet as defined in USB core specification. Includes sync, PID, eop, etc.
		144	Total

The exact details of the function ($f(x)$) are up to the particular implementation. However, it should be obvious that the goal is to minimize the area under the curve between the approximation function and the *Last Start* curve, without dipping below the *LastStart* line, while at the same time keeping the check as simple as possible for hardware implementation. The $f(x)$ in [Figure 56, “Best Fit Approximation” on page 421](#) was constructed using the following pseudo-code test on each transaction size data point. This algorithm assumes that the host controller keeps track of the remaining bits in the frame.

```
Algorithm CheckTransactionWillFit (MaximumPacketSize, HC_BytesLeftInFrame)
```

```
Begin
```

```
    Local Temp = MaximumPacketSize + 192
```

```
    Local rvalue = TRUE
```

```
    If MaximumPacketSize >= 128 then
```

```
        Temp += 128
```

```
    End If
```

```
    If Temp > HC_BytesLeftInFrame then
```

```
        Rvalue = FALSE
```

```
    End If
```

```
    Return rvalue
```

```
End
```

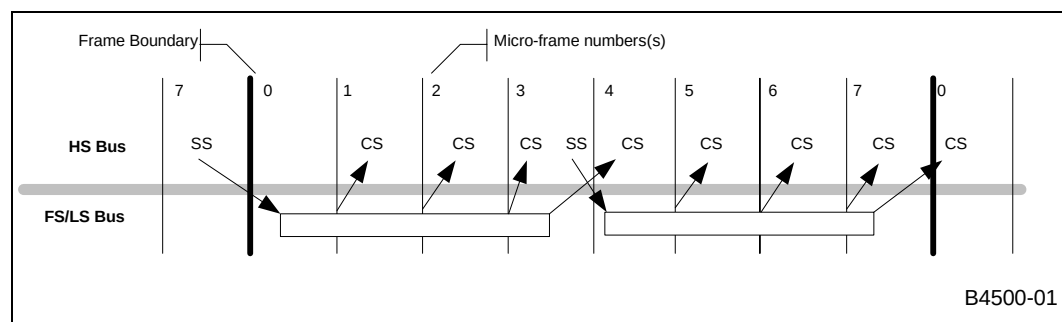
This algorithm takes two inputs, the current maximum packet size of the transaction and a hardware counter of the number of bytes left in the current micro-frame. It unconditionally adds a simple constant of 192 to the maximum packet size to account for a first-order effect of transaction overhead and bit stuffing. If the transaction size is greater than or equal to 128 bytes, then an additional constant of 128 is added to the running sum to account for the additional worst-case bit stuffing of payloads larger than 128. An inflection point was inserted at 128 because the $f(x)$ plot was getting close to the *LastStart* line.



9.14.5 Periodic Schedule Frame Boundaries versus Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(s) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in [Figure 57, “Frame Boundary Relationship between HS bus and FS/LS Bus” on page 423](#)). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

Figure 57. Frame Boundary Relationship between HS bus and FS/LS Bus



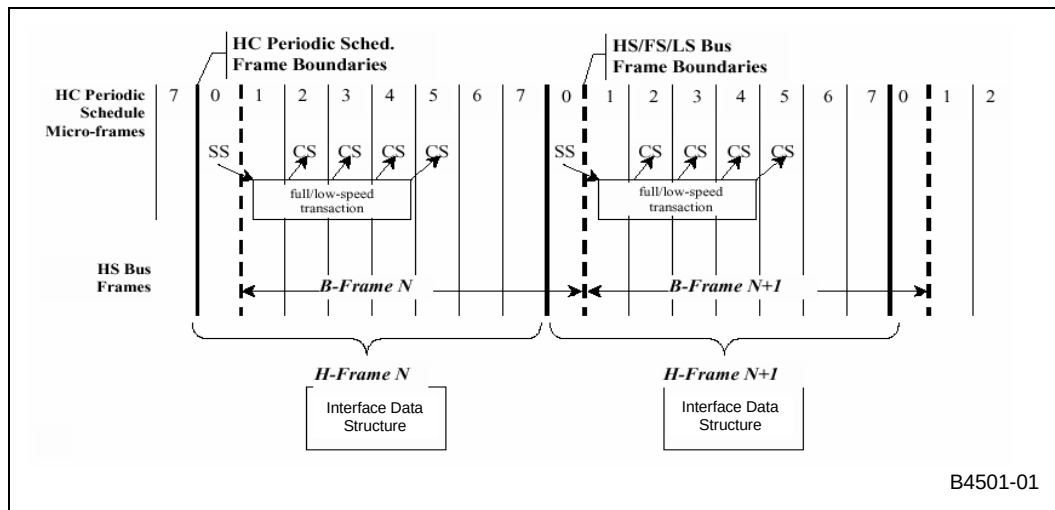
The simple projection, as [Figure 57, “Frame Boundary Relationship between HS bus and FS/LS Bus” on page 423](#) illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX) documented in [Section 9.12.4, “FRINDEX” on page 378](#) and initially illustrated in [Section 9.14.4, “Schedule Traversal Rules” on page 418](#). Bits FRINDEX[2:0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13:3]. Both FRINDEX[13:3] and the SOF value are incremented based on FRINDEX[2:0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields host controller periodic schedule and bus frame boundary relationship as illustrated in [Figure 58, “Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries” on page 424](#). This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full- and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface. The reasons for selecting this phase-shift are beyond the scope of this specification.

[Figure 58, “Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries” on page 424](#) illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the 1-millisecond boundaries is called *H-Frames*. The high-speed bus's view of the 1-ms boundaries is called *B-Frames*.



Figure 58. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries



H-Frame boundaries for the host controller correspond to increments of FRINDEX[13:3]. Micro-frame numbers for the *H-Frame* are tracked by FRINDEX[2:0]. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (i.e. the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (i.e. B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 58, "Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries" on page 424. The host controller's periodic schedule is naturally aligned to H-Frames. Software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in Section 9.12.4, "FRINDEX" on page 378, the SOF Value can be implemented as a shadow register (in this example, called SOFV), which lags the FRINDEX register bits [13:3] by one micro-frame count. Table 172, "Operation of FRINDEX and SOFV (SOF Value Register)" on page 424 illustrates the required relationship between the value of FRINDEX and the value of SOFV. This lag behavior can be accomplished by incrementing FRINDEX[13:3] based on carry-out on the 7 to 0 increment of FRINDEX[2:0] and incrementing SOFV based on the transition of 0 to 1 of FRINDEX[2:0].

Software is allowed to write to FRINDEX. Section 9.12.4, "FRINDEX" on page 378 provides the requirements that software should adhere when writing a new value in FRINDEX.

Table 172. Operation of FRINDEX and SOFV (SOF Value Register) (Sheet 1 of 2)

	Current			Next	
FRINDEX[F]	SOFV	FRINDEX[?F]	FRINDEX[F]	SOFV	FRINDEX[?F]
N	N	111b	N+1	N	000b
N+1	N	000b	N+1	N+1	001b
N+1	N+1	001b	N+1	N+1	010b
N+1	N+1	010b	N+1	N+1	011b
N+1	N+1	011b	N+1	N+1	100b
Note: Where [F] = [13:3]; [μF] = [2:0]					

**Table 172. Operation of FRINDEX and SOFV (SOF Value Register) (Sheet 2 of 2)**

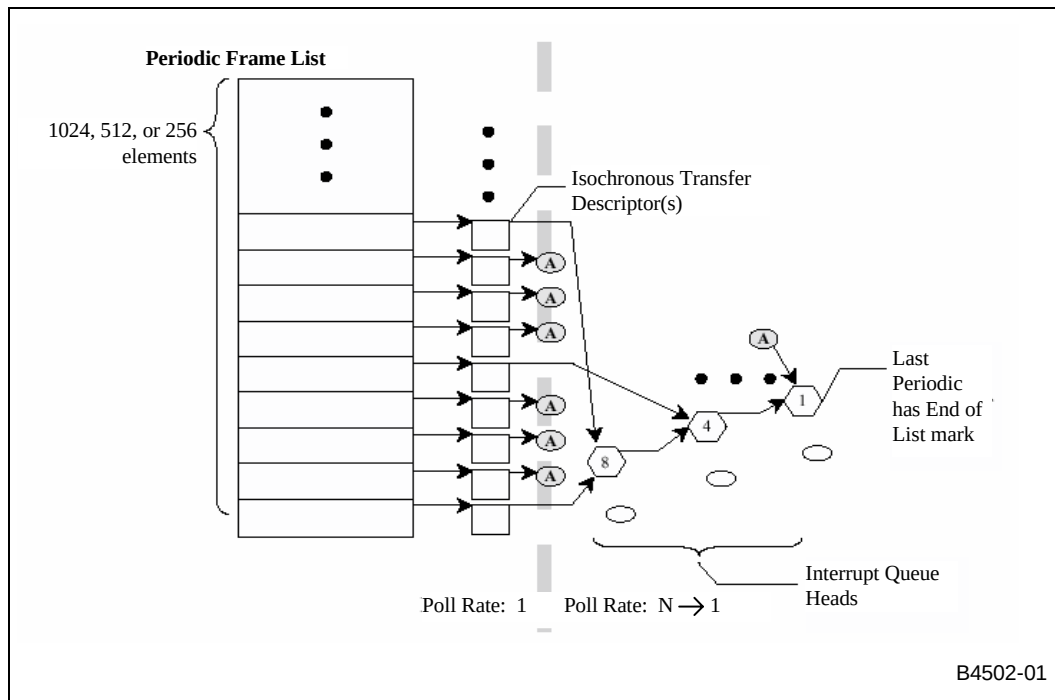
	Current			Next	
N+1	N+1	100b	N+1	N+1	101b
N+1	N+1	101b	N+1	N+1	110b
N+1	N+1	110b	N+1	N+1	111b
Note: Where [F] = [13:3]; [μF] = [2:0]					

9.14.6 Periodic Schedule

The periodic schedule traversal is enabled or disabled via the *Periodic Schedule Enable* bit in the USBCMD register. If the *Periodic Schedule Enable* bit is set to a zero, then the host controller simply does not try to access the periodic frame list via the *PERIODICLISTBASE* register. Likewise, when the *Periodic Schedule Enable* bit is a one, then the host controller does use the *PERIODICLISTBASE* register to traverse the periodic schedule. The host controller will not react to modifications to the *Periodic Schedule Enable* immediately. In order to eliminate conflicts with split transactions, the host controller evaluates the *Periodic Schedule Enable* bit only when FRINDEX[2:0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 000b micro-frame. These work items must be removed from the schedule before the *Periodic Schedule Enable* bit is written to a zero. The *Periodic Schedule Status* bit in the USBSTS register indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by writing a one (or zero) to the *Periodic Schedule Enable* bit in the USBCMD register. Software then can poll the *Periodic Schedule Status* bit to determine when the periodic schedule has made the desired transition. Software must not modify the *Periodic Schedule Enable* bit unless the value of the *Periodic Schedule Enable* bit equals that of the *Periodic Schedule Status* bit.

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. [Figure 59, “Example Periodic Schedule” on page 426](#) illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (e.g. closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

Figure 59. Example Periodic Schedule



9.14.7 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in [Section 9.13.3, “Isochronous \(High-Speed\) Transfer Descriptor \(iTID\)”](#) on page 391. There are four distinct sections to an iTD:

- The first field is the *Next Link Pointer*. This field is for schedule linkage purposes only;
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

9.14.7.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits [12:3] to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, which map directly to FRINDEX register bits [2:0]. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits [2:0] to index into the transaction description array. If the *active* bit in the *Status* field of the indexed transaction description is set to zero, the host controller ignores the iTD and follows the *Next* pointer to the next schedule data structure.



When the indexed *active* bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum packet size, etc.). It also uses the *Page Select* (PG) field to index the *buffer pointer* array, storing the selected *buffer pointer* and the next sequential *buffer pointer*. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's *Transaction Offset* field. The host controller uses the endpoint addressing information and *I/O-bit* to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the *Status* field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 00B) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the *Maximum Packet Size* field. An iTD supports high-bandwidth pipes via the *Mult* (multiplier) field. When the *Mult* field is 1, 2, or 3, the host controller executes the specified number of *Maximum Packet* sized bus transactions for the endpoint in the current micro-frame. In other words, the *Mult* field represents a transaction count for the endpoint in the current micro-frame. If the *Mult* field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the *Mult* field.

For OUT transfers, the value of the *Transaction X Length* field represents the total bytes to be sent during the micro-frame. The *Mult* field must be set by software to be consistent with *Transaction X Length* and *Maximum Packet Size*. The host controller will send the bytes in *Maximum Packet Size*'d portions. After each transaction, the host controller decrements its local copy of *Transaction X Length* by *Maximum Packet Size*. The number of bytes the host controller sends is always *Maximum Packet Size* or *Transaction X Length*, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3 x 1024 bytes.

For IN transfers, the host controller issues *Mult* transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use *Maximum Packet Size* to detect packet babble errors. The host controller keeps the sum of bytes received in the *Transaction X Length* field. After all transactions for the endpoint have completed for the micro-frame, *Transaction X Length* contains the total bytes received. If the final value of *Transaction X Length* is less than the value of *Maximum Packet Size*, then less data than was allowed for was received from the associated endpoint. This *short packet* condition does not set the *USBINT* bit in the USBSTS register to a one. The host controller will not detect this condition. If the device sends more than *Transaction X Length* or *Maximum Packet Size* bytes (whichever is less), then the host controller will set the *Babble Detected* bit to a one and set the *Active* bit to a zero. Note, that the host controller is not required to update the iTD field *Transaction X Length* in this error scenario. If the *Mult* field is greater than one, then the host controller will automatically execute the value of *Mult* transactions. The host controller will not execute all *Mult* transactions if:

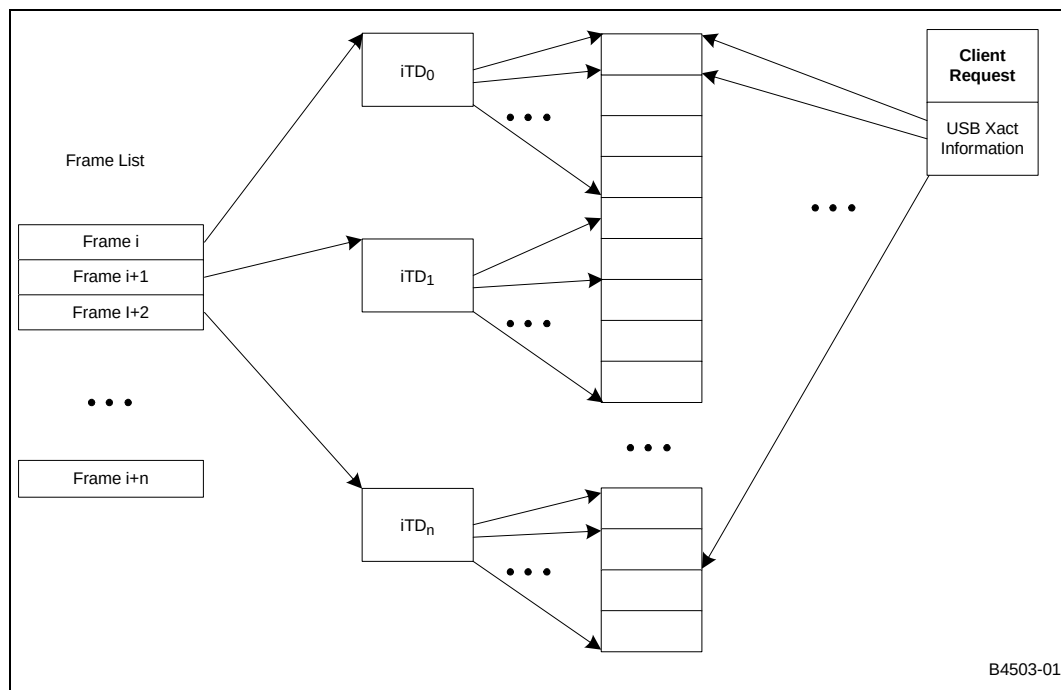


- The endpoint is an OUT and *Transaction X Length* goes to zero before all the *Mult* transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before *Mult* transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame. Refer to Appendix D for a table summary of the host controller required behavior for all the high-bandwidth transaction cases.

9.14.7.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 60, “Example Association of iTDs to Client Request Buffer” on page 429 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (i.e. the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller's frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.


Figure 60. Example Association of iTDs to Client Request Buffer


B4503-01

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the *PG* field is set to index the correct physical buffer page pointer and the *Transaction Offset* field is set relative to the correct buffer pointer page (e.g. the same one referenced by the *PG* field). When the host controller executes a transaction it selects a transaction description record based on *FRINDEX[2:0]*. It then uses the current *Page Buffer Pointer* (as selected by the *PG* field) and concatenates to the *transaction offset* field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available *Page Buffer Pointer*. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap a page boundary. Doing so will yield undefined behavior. The host controller hardware is not required to 'alias' the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

9.14.7.2.1 Periodic Scheduling Threshold

The *Isochronous Scheduling Threshold* field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by system software when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.



The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty-factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the *Isochronous Scheduling Threshold* field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but will always dump any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the *Isochronous Scheduling Threshold* field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the *current* micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the *Isochronous Scheduling Threshold* field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the *Isochronous Scheduling Threshold* field. For example, if the count value were 2, then the host controller keeps a *window* of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

9.14.8 Asynchronous Schedule

The Asynchronous schedule traversal is enabled or disabled via the *Asynchronous Schedule Enable* bit in the USBCMD register. If the *Asynchronous Schedule Enable* bit is set to a zero, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, when the *Asynchronous Schedule Enable* bit is a one, then the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the *Asynchronous Schedule Enable* bit are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

The *Asynchronous Schedule Status* bit in the USBSTS register indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to the *Asynchronous Schedule Enable* bit in the USBCMD register. Software then can poll the *Asynchronous Schedule Status* bit to



determine when the asynchronous schedule has made the desired transition. Software must not modify the *Asynchronous Schedule Enable* bit unless the value of the *Asynchronous Schedule Enable* bit equals that of the *Asynchronous Schedule Status* bit.

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the *ASYNCLISTADDR* register. The default value of the *ASYNCLISTADDR* register after reset is undefined and the schedule is disabled when the *Asynchronous Schedule Enable* bit is a zero.

Software may only write this register with defined results when the schedule is disabled, for example, *Asynchronous Schedule Enable* bit in the *USBCMD* and the *Asynchronous Schedule Status* bit in the *USBSTS* register are zero. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting the *Asynchronous Schedule Enable* bit is set to one. The asynchronous schedule is actually enabled when the *Asynchronous Schedule Status* bit is a one.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the *ASYNCLISTADDR* register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller “completes” processing the asynchronous schedule, it retains the value of the last accessed queue head’s horizontal pointer in the *ASYNCLISTADDR* register. Next time the asynchronous schedule is accessed, this is the first data structure that will be serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller “completes” processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition (i.e. see [Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435](#))
- The schedule has been disabled via the *Asynchronous Schedule Enable* bit in the *USBCMD* register.

The queue heads in the asynchronous list are linked into a simple circular list as shown in [Figure 55, “General Format of Asynchronous Schedule List” on page 419](#). Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

9.14.8.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. System software writes the physical memory address of a queue head into the *ASYNCLISTADDR* register, then enables the list by setting the *Asynchronous Schedule Enable* bit in the *USBCMD* register to a one.



When inserting a queue head into an active list, software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have *T-Bits* set to a one or reference valid qTDs and the *Horizontal Pointer* references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)

--
-- Requirement: all inputs must be properly initialized.
--
-- pQHeadCurrent is a pointer to a queue head that is
-- already in the active list
-- pQHeadNew is a pointer to the queue head to be added
--
-- This algorithm links a new queue head into a existing
-- list
--

pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer

pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)

End InsertQueueHead
```

9.14.8.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. Software deactivates the asynchronous schedule by setting the *Asynchronous Schedule Enable* bit in the USBCMD register to a zero. Software can determine when the list is idle when the *Asynchronous Schedule Status* bit in the USBSTS register is a zero. The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list via the following algorithm. As illustrated, the unlinking is quite easy. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.



```

UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)

--
-- Requirement: all inputs must be properly initialized.
--
-- pQHeadPrevious is a pointer to a queue head that
-- references the queue head to remove
-- pQHeadToUnlink is a pointer to the queue head to be
-- removed
-- pQHeadNext is a pointer to a queue head still in the
-- schedule. Software provides this pointer with the
-- following strict rules:
-- if the host software is one queue head, then
-- pQHeadNext must be the same as
-- QueueheadToUnlink.HorizontalPointer. If the host
-- software is unlinking a consecutive series of
-- queue heads, QHeadNext must be set by software to
-- the queue head remaining in the schedule.
--
-- This algorithm unlinks a queue head from a circular list
--

pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
End UnlinkQueueHead

```

If software removes the queue head with the *H-bit* set to a one, it must select another queue head still linked into the schedule and set its *H-bit* to a one. This should be completed before removing the queue head. The requirement is that software keep one queue head in the asynchronous schedule, with its *H-bit* set to a one. At the point software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers, etc.). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple,

lightweight handshake that is used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

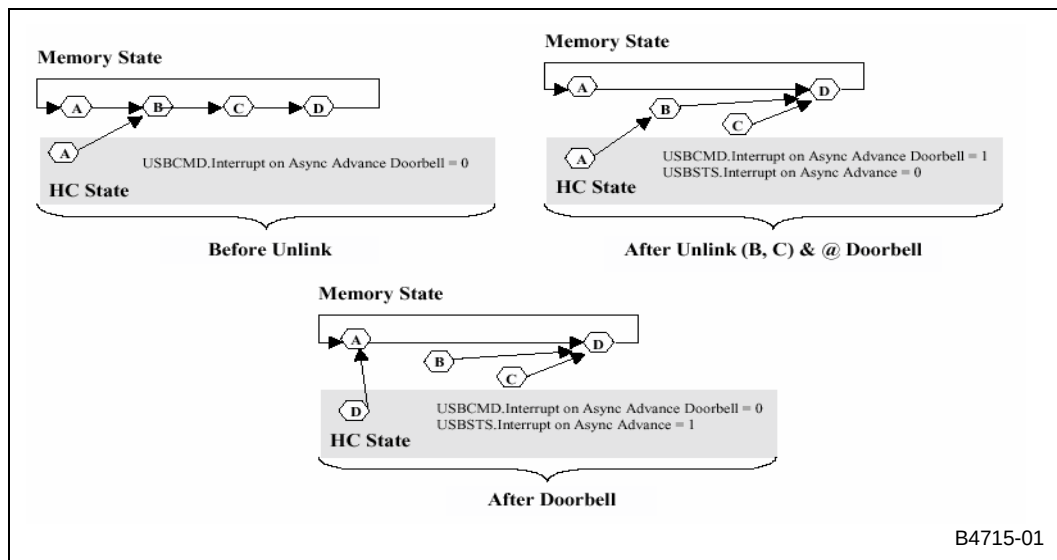
The handshake is implemented with three bits in the host controller. The first bit is a command bit (*Interrupt on Async Advance Doorbell* bit in the USBCMD register) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (*Interrupt on Async Advance* bit in the USBSTS register) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit to a one, it also sets the command bit to a zero. The third bit is an interrupt enable (*Interrupt on Async Advance* bit in the USBINTR register) that is matched with the status bit. If the status bit is a one and the interrupt enable bit is a one, then the host controller will assert a hardware interrupt.

Figure 61, “Generic Queue Head Unlink Scenario” on page 434 illustrates a general example. In this example, consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that will remain in the asynchronous schedule.

When the host controller observes that doorbell bit being set to a one, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (i.e. traversed beyond queue head (B) in this example).

Figure 61. Generic Queue Head Unlink Scenario



Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue [twice]) before setting the *Advance on Async* status bit to a one.



Software may re-use the memory associated with the removed queue heads after it observes the *Interrupt on Async Advance* status bit is set to a one, following assertion of the doorbell. Software should acknowledge the *Interrupt on Async Advance* status as indicated in the USBSTS register, before using the doorbell handshake again.

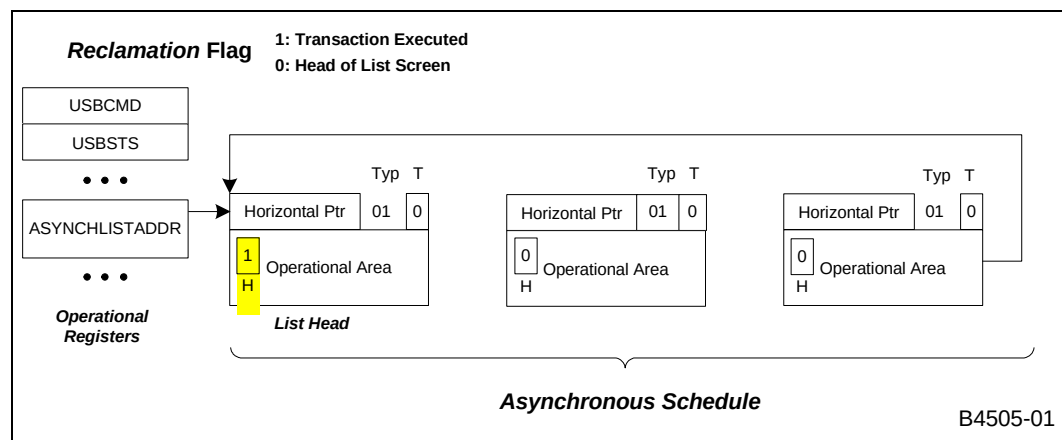
9.14.8.3 Empty Asynchronous Schedule Detection

The Enhanced Host Controller Interface uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see [Figure 50, “Queue Head Structure Layout” on page 404](#)) defines an *H-bit* in the queue head, which allows software to mark a queue head as being the *head* of the reclaim list. The Enhanced Host Controller Interface also keeps a 1-bit flag in the USBSTS register (*Reclamation*) that is set to a zero when the Enhanced Interface Host Controller observes a queue head with the H-bit set to a one. The reclamation flag in the status register is set to one when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see [“Asynchronous Schedule Traversal: Start Event” on page 438](#)).

If the Enhanced Host Controller Interface ever encounters an *H-bit* of one and a *Reclamation* bit of zero, the EHCI controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is illustrated in [Figure 62](#).

Figure 62. Asynchronous Schedule List with Annotation to Mark Head of List



Software must ensure there is at most one queue head with the *H-bit* set to a one, and that it is always coherent with respect to the schedule.

9.14.8.4 Restarting Asynchronous Schedule Before EOF

There are many situations where the host controller will detect an empty list *long* before the end of the micro-frame. It is important to remember that under many circumstances the schedule traversal has stopped due to Nak/Nyet responses from all endpoints.

An example of particular interest is when a start-split for a bulk endpoint occurs early in the micro-frame. Given the EHCI simple traversal rules, the complete-split for that transaction may Nak/Nyet out very quickly. If it is the only item in the schedule, then the host controller will cease traversal of the Asynchronous schedule very early in the micro-frame. In order to provide reasonable service to this endpoint, the host controller should issue the complete-split before the end of the current micro-frame, instead of waiting until the next micro-frame. When the reason for host controller idling

asynchronous schedule traversal is because of empty list detection, it is mandatory the host controller implement a 'waking' method to resume traversal of the asynchronous schedule. An example method is described below.

9.14.8.4.1 Example Method for Restarting Asynchronous Schedule Traversal

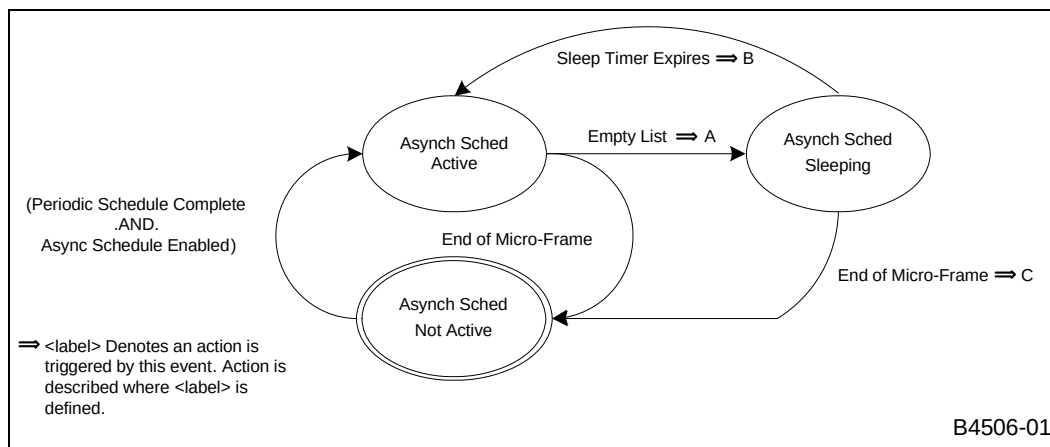
The reason for idling the host controller when the list is empty is to keep the host controller from unnecessarily occupying too much memory bandwidth. The question is: *how long should the host controller stay idle before restarting?*

The answer in this example is based on deriving a manifest constant, which is the amount of time the host controller will stay idle before restarting traversal. In this example, the manifest constant is called *AsyncSchedSleepTime*, and has a value of 10μsec. The value is derived based on the analysis in “[Example Derivation for AsyncSchedSleepTime](#)” on page 437 The traversal algorithm is simple:

- Traverse the Asynchronous schedule until the either an End-Of-micro-Frame event occurs, or an empty list is detected. If the event is an End-of-micro-Frame, go attempt to traverse the Periodic schedule. If the event is an empty list, then set a sleep timer and go to a *schedule sleep* state.
- When the sleep timer expires, set working context to the Asynchronous Schedule start condition and go to *schedule active* state. The start context allows the HC to reload *Nakcnt* fields, etc. so the HC has a chance to run for more than one iteration through the schedule.

This process simply repeats itself each micro-frame. [Figure 63](#) illustrates a sample state machine to manage the active and sleep states of the Asynchronous Schedule traversal policy. There are three states: Actively traversing the Asynchronous schedule, Sleeping, and Not Active. The last two are similar in terms of interaction with the Asynchronous schedule, but the Not Active state means that the host controller is busy with the Periodic schedule or the Asynchronous schedule is not enabled. The Sleeping state is specifically a special state where the host controller is just waiting for a period of time before resuming execution of the Asynchronous schedule.

Figure 63. Example State Machine for Managing Asynchronous Schedule Traversal



The actions referred to in [Figure 63](#) are defined in [Table 173](#).

**Table 173. Asynchronous Schedule State Machine Transition Actions**

Action	Action Description Label
A	On detection of the empty list, the host controller sets the <i>AsynchronousTraversalSleepTimer</i> to <i>AsyncSchedSleepTime</i> .
B	When the <i>AsynchronousTraversalSleepTimer</i> expires, the host controller sets the <i>Reclamation</i> bit in the USBSTS register to a one and moves the Nak Counter reload state machine to <i>WaitForListHead</i> .
C	The host controller cancels the sleep timer (<i>AsynchronousTraversalSleepTimer</i>).

Async Sched Not Active

This is the initial state of the traversal state machine after a host controller reset. The traversal state machine will not leave this state when the *Asynchronous Schedule Enable* bit in the USBCMD register is a zero.

This state is entered from **Async Sched Active** or **Async Sched Sleeping** states when the end-of-micro-frame event is detected.

Async Sched Active

This state is entered from the **Async Sched Not Active** state when the periodic schedule is not active. It is also entered from the **Async Sched Sleeping** states when the *AsynchronousTraversalSleepTimer* expires. On every transition into this state, the host controller sets the *Reclamation* bit in the USBSTS register to a one.

While in this state, the host controller will continually traverse the asynchronous schedule until either the end of micro-frame or an empty list condition is detected.

Async Sched Sleeping

The state is entered from the **Async Sched Active** state when a schedule empty condition is detected. On entry to this state, the host controller sets the *AsynchronousTraversalSleepTimer* to *AsyncSchedSleepTime*.

9.14.8.4.2 Example Derivation for AsyncSchedSleepTime

The derivation is based on analysis of what work the host controller could be doing next. It assumes the host controller does not keep any state about what work is possibly pending in the asynchronous schedule. The schedule could contain any mix of the possible combinations of high- full- or low-speed control and bulk requests. [Table 174, "Typical Low- /Full-Speed Transaction Times" on page 437](#) summarizes some of the typical 'next transactions' that could be in the schedule, and the amount of time (e.g. *footprint*, or *wall clock*) the transaction will take to complete.

Table 174. Typical Low- /Full-Speed Transaction Times (Sheet 1 of 2)

Transaction Attributes		Footprint (Time)	Description
Speed	HS	11.9 μ s 9.45 μ s	Maximum foot print for a worst-case, full-sized bulk data transaction.
Size	512		Maximum footprint for an approximate best-case, full-sized bulk data transaction.
Type	Bulk		
Speed	FS	~ 50 μ s	Approximate typical for full-sized bulk data. An 8-byte low-speed is about 2x, or between 90 and 100 μ s.
Size	64		
Type	Bulk		



Table 174. Typical Low- /Full-Speed Transaction Times (Sheet 2 of 2)

Transaction Attributes		Footprint (Time)	Description
Speed	FS	~12 μ s	Approximate typical for 8-byte bulk/control (i.e. setup)
Size	8		
Type	Cntrl		

An *AsyncSchedSleepTime* value of 10 μ s provides a reasonable relaxation of the system memory load and still provides a good level of service for the various transfer types and payload sizes. For example, say we detect an empty list after issuing a start-split for a 64-byte full-speed bulk request. Assuming this is the only thing in the list, the host controller will get the results of the full-speed transaction from the hub during the fifth complete-split request. If the full-speed transaction was an IN and it nak'd, the 10 μ s sleep period would allow the host controller to get the NAK results on the first complete-split.

9.14.8.5 Asynchronous Schedule Traversal: Start Event

Once the HC has *idled* itself via the empty schedule detection (Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435), it will naturally *activate* and begin processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the HC must occasionally *re-activate* during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. The requirements and method for this restart are described in Section 9.14.8.4, “Restarting Asynchronous Schedule Before EOF” on page 435. Asynchronous schedule *Start Events* are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state (see Section 9.14.8.4, “Restarting Asynchronous Schedule Before EOF” on page 435).

9.14.8.6 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature (Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435) depends on the proper management of the *Reclamation* bit in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head while traversing the asynchronous schedule (See Section 9.14.10.1, “Fetch Queue Head” on page 443).

It is required that the host controller sets the *Reclamation* bit to a one whenever an asynchronous schedule traversal *Start Event*, as documented in “Asynchronous Schedule Traversal: Start Event” on page 438, occurs. The *Reclamation* bit is also set to a one whenever the host controller executes a transaction while traversing the asynchronous schedule (see Section 9.14.10.3, “Execute Transaction” on page 444). The host controller sets the *Reclamation* bit to a zero whenever it finds a queue head with its *H-bit* set to a one. Software should only set a queue head's *H-bit* if the queue head is in the asynchronous schedule. If software sets the *H-bit* in an interrupt queue head to a one, the resulting behavior is undefined. The host controller may set the *Reclamation* bit to a zero when executing from the periodic schedule.



9.14.9 Operational Model for Nak Counter

This section describes the operational model for the *NakCnt* field defined in a queue head (see [Section 9.13.6, “Queue Head” on page 404](#)). Software should not use this feature for interrupt queue heads. This rule is not required to be enforced by the host controller.

USB protocol has built-in flow control via the Nak response by a device. There are several scenarios, beyond the Ping feature, where an endpoint may naturally Nak or Nyet the majority of the time. An example is the host controller management of the split transaction protocol for control and bulk endpoints. All bulk endpoints (High- or Full-speed) are serviced via the same asynchronous schedule. The time between the *Start-split* transaction and the first *Complete-split* transaction could be very short (i.e. like when the endpoint is the only one in the asynchronous schedule). The hub NYETs (effectively Naks) the *Complete-split* transaction until the classic transaction is complete. This could result in the host controller thrashing memory, repeatedly fetching the queue head and executing the transaction to the hub, which will not complete until after the transaction on the classic bus completes.

There are two component fields in a queue head to support the throttling feature: a counter field (*NakCnt*), and a counter reload field (*RL*). *NakCnt* is used by the host controller as one of the criteria to determine whether or not to execute a transaction to the endpoint. There are two operational modes associated with this counter:

- **Not Used.** This mode is set when the *RL* field is zero. The host controller ignores the *NakCnt* field for any execution of transactions through a queue head with an *RL* field of zero. Software must use this selection for interrupt endpoints.
- **Nak Throttle Mode.** This mode is selected when the *RL* field is non-zero. In this mode, the value in the *NakCnt* field represents the maximum number of Nak or Nyet responses the host controller will tolerate on each endpoint. In this mode, the HC will decrement the *NakCnt* field based on the token/handshake criteria listed in [Table 175, “NakCnt Field Adjustment Rules” on page 439](#). The host controller must reload *NakCnt* when the endpoint successfully moves data (e.g. policy to reward device for moving data).

Table 175. NakCnt Field Adjustment Rules

Token	Handshake	
	Handshake NAK	NYET
IN/PING	decrement <i>NakCnt</i>	N/A (protocol error)
OUT	decrement <i>NakCnt</i>	No Action ¹ Start
Split	decrement <i>NakCnt</i>	N/A (protocol error)
Complete Split	No Action	Decrement <i>NakCnt</i>
Notes: 1. Recommended behavior on this response is to reload <i>NakCnt</i> .		

In summary, system software enables the counter by setting the reload field (*RL*) to a non-zero value. The host controller may execute a transaction if *NakCnt* is non-zero. The host controller will not execute a transaction if *NakCnt* is zero. The reload mechanism is described in detail in [Section 9.14.9.1, “Nak Count Reload Control” on page 440](#).

Note that when all queue heads in the Asynchronous Schedule either exhausts all transfers or all *NakCnt*'s go to zero, then the host controller will detect an empty Asynchronous Schedule and idle schedule traversal (see [Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435](#)).



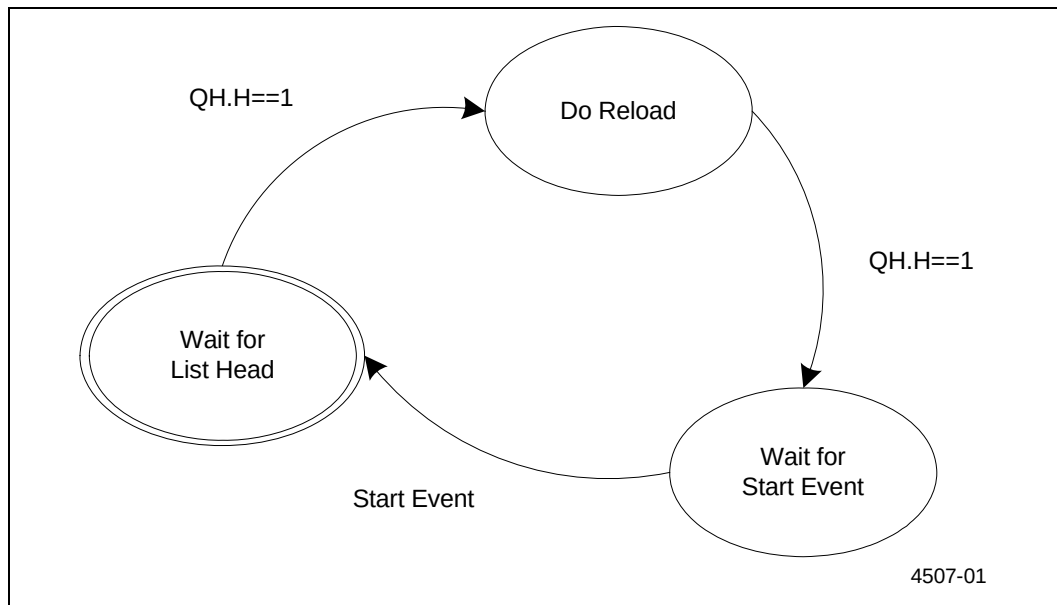
Any time the host controller begins a new traversal of the Asynchronous Schedule, a *Start Event* is assumed, see [“Asynchronous Schedule Traversal: Start Event” on page 438](#). Every time a Start-Event occurs, the Nak Count reload procedure is enabled.

9.14.9.1 Nak Count Reload Control

When the host controller reaches the *Execute Transaction* state for a queue head (meaning that it has an active operational state), it checks to determine whether the *NakCnt* field should be reloaded from *RL* (see [Section 9.14.10.3, “Execute Transaction” on page 444](#)). If the answer is yes, then *RL* is copied into *NakCnt*. After the reload or if the reload is not active, the host controller evaluates whether to execute the transaction.

The host controller must reload nak counters (*NakCnt* see [Figure 50, “Queue Head Structure Layout” on page 404](#)) in queue heads during the first pass through the reclamation list after an asynchronous schedule Start Event (see [“Asynchronous Schedule Traversal: Start Event” on page 438](#) for the definition of the Start Event). The Asynchronous Schedule should have at most one queue head marked as the head (see [Figure 62 on page 435](#)). [Figure 64](#) illustrates an example state machine that satisfies the operational requirements of the host controller detecting the first pass through the Asynchronous Schedule. This state machine is maintained internal to the host controller and is only used to gate reloading of the nak counter during the queue head traversal state: *Execute Transaction* ([Figure 65 on page 442](#)). The host controller does not perform the nak counter reload operation if the *RL* field (see [Figure 50, “Queue Head Structure Layout” on page 404](#)) is set to zero.

Figure 64. Example HC State Machine for Controlling Nak Counter Reloads



9.14.9.1.1 Wait for List Head

This is the initial state. The state machine enters this state from **Wait for Start Event** when a start event as defined in [“Asynchronous Schedule Traversal: Start Event” on page 438](#) occurs. The purpose of this state is to wait for the first observation of the head of the Asynchronous Schedule. This occurs when the host controller fetches a queue head whose *H-bit* is set to a one.



9.14.9.1.2 Do Reload

This state is entered from the **Wait for List Head** state when the host controller fetches a queue head with the *H-bit* set to a one. While in this state, the host controller will perform nak counter reloads for every queue head visited that has a non-zero nak reload value (*RL*) field.

9.14.9.1.3 Wait for Start Event

This state is entered from the *Do Reload* state when a queue head with the *H-bit* set to a one is fetched. While in this state, the host controller will not perform nak counter reloads.

9.14.10 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the Queue Element Transfer Descriptor (qTD) structure defined in [Section 9.13.5, “Queue Element Transfer Descriptor \(qTD\)” on page 399](#). One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed (see Overlay area defined in [Figure 50, “Queue Head Structure Layout” on page 404](#)). Each qTD represents one or more bus transactions, which is defined in the context of this specification as a *transfer*.

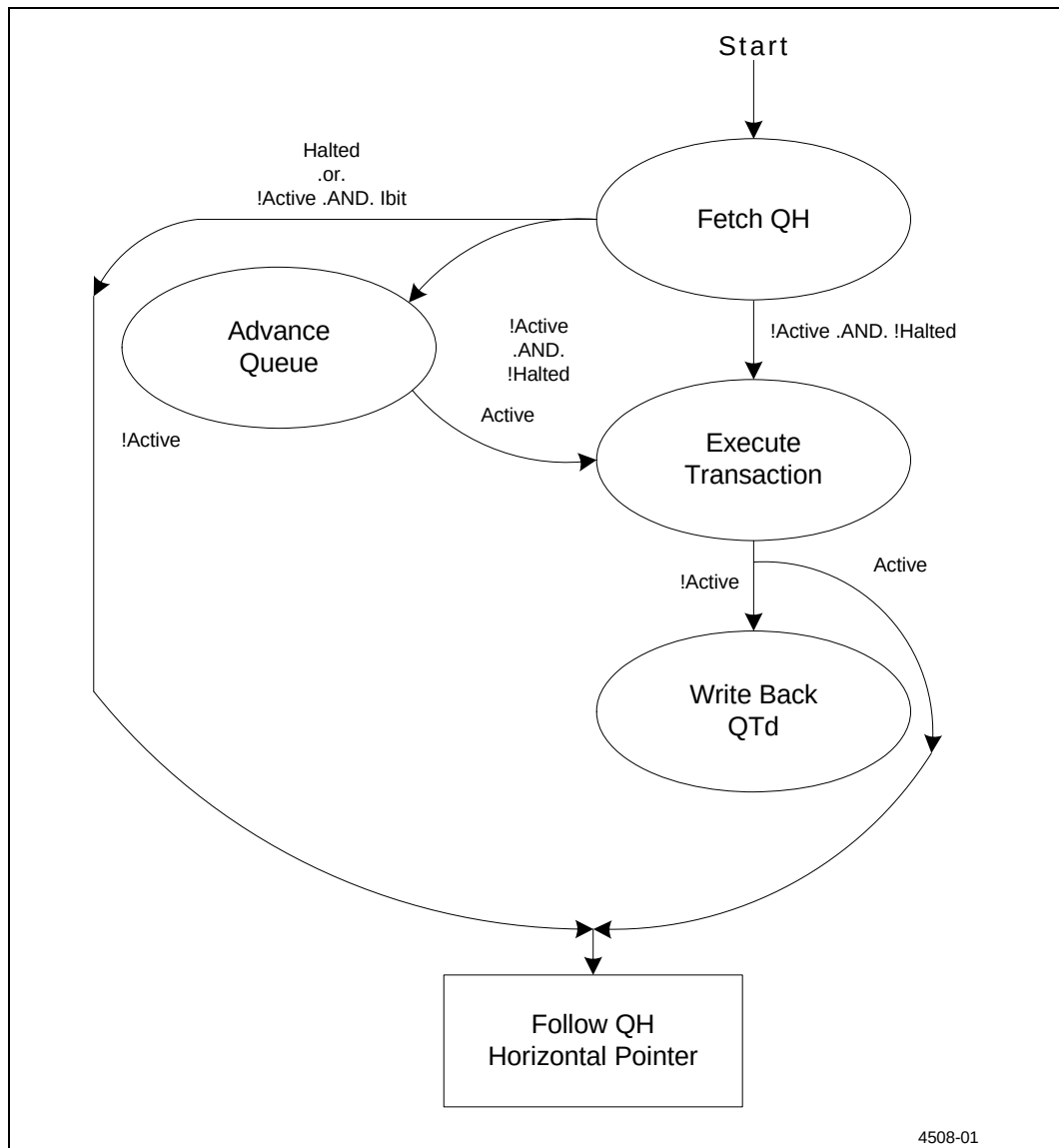
The general processing model for the host controller's use of a queue head is simple:

- read a queue head,
- execute a transaction from the overlay area,
- write back the results of the transaction to the overlay area
- move to the next queue head.

If the host controller encounters errors during a transaction, the host controller will set one (or more) of the error reporting bits in the queue head's *Status* field. The *Status* field accumulates all errors encountered during the execution of a qTD (e.g. the error bits in the queue head *Status* field are 'sticky' until the transfer (qTD) has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (e.g. buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions will occur for the endpoint and the host controller will not advance the queue.

An example host controller operational state machine of a queue head traversal is illustrated in [Figure 65](#). This state machine is a model for how a host controller should traverse a queue head. The host controller must be able to advance the queue from the *Fetch QH* state in order to avoid all hardware/software race conditions. This simple mechanism allows software to simply link qTDs to the queue head and *activate* them, then the host controller will always *find* them if/when they are reachable.

Figure 65. Host Controller Queue Head Traversal State Machine



This traversal state machine applies to all queue heads, regardless of transfer type or whether split transactions are required. The following sections describe each state. Each state description describes the entry criteria. The **Execute Transaction** state (Section 9.14.10.3, “Execute Transaction” on page 444) describes the basic requirements for all endpoints. Section 9.14.12.1, “Split Transactions for Asynchronous Transfers” on page 453 and Section 9.14.12.2, “Split Transaction Interrupt” on page 455 describe details of the required extensions to the **Execute Transaction** state for endpoints requiring split transactions.

Note: Prior to software placing a queue head into either the periodic or asynchronous list, software must ensure the queue head is properly initialized. Minimally, the queue head should be initialized to the following:

- Valid static endpoint state



- For the very first use of a queue head, software may zero-out the queue head transfer overlay, then set the *Next qTD Pointer* field value to reference a valid qTD.

9.14.10.1 Fetch Queue Head

A queue head can be referenced from the physical address stored in the ASYNCLISTADDR Register (Section 9.12.7, “ASYNCLISTADDR; ENDPOINTLISTADDR” on page 380). Additionally, it may be referenced from the *Next Link Pointer* field of an iTD, siTD, FSTN or another Queue Head. If the referencing link pointer has the *Typ* field set to indicate a queue head, it is assumed to reference a queue head structure as defined in Figure 50, “Queue Head Structure Layout” on page 404.

While in this state, the host controller performs operations to implement empty schedule detection (Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435) and Nak Counter reloads (Section 9.14.9, “Operational Model for Nak Counter” on page 439). After the queue head has been fetched, the host controller conducts the following queries for empty schedule detection:

- If queue head is not an interrupt queue head (i.e. *S-mask* is a zero), and
- The *H-bit* is a one, and
- The *Reclamation* bit in the USBSTS register is a zero.

When these criteria are met, the host controller will stop traversing the asynchronous list (as described in Section 9.14.8.3, “Empty Asynchronous Schedule Detection” on page 435). When the criteria are not met, the host controller continues schedule traversal. If the queue head is not an interrupt and the *H-bit* is a one and the *Reclamation* bit is a one, then the host controller sets the *Reclamation* bit in the USBSTS register to a zero before completing this state. The operations for reloading of the Nak Counter are described in detail in Section 9.14.9, “Operational Model for Nak Counter” on page 439.

This state is complete when the queue head has been read on-chip.

9.14.10.2 Advance Queue

To advance the queue, the host controller must find the next qTD, adjust pointers, perform the overlay and write back the results to the queue head.

This state is entered from the **FetchQHD** state if the overlay *Active* and *Halt* bits are set to zero. On entry to this state, the host controller determines which next pointer to use to fetch a qTD, fetches a qTD and determines whether or not to perform an overlay. Note that if the *I-bit* is a one and the *Active* bit is a zero, the host controller immediately skips processing of this queue head, exits this state and uses the horizontal pointer to the next schedule data structure. If the field *Bytes to Transfer* is not zero and the *T-bit* in the *Alternate Next qTD Pointer* is set to zero, then the host controller uses the *Alternate Next qTD Pointer*. Otherwise, the host controller uses the *Next qTD Pointer*. If *Next qTD Pointer's T-bit* is set to a one, then the host controller exits this state and uses the horizontal pointer to the next schedule data structure.

Using the selected pointer the host controller fetches the referenced qTD. If the fetched qTD has its *Active* bit set to a one, the host controller moves the pointer value used to reach the qTD (*Next* or *Alternate Next*) to the *Current qTD Pointer* field, then performs the overlay. If the fetched qTD has its *Active* bit set to a zero, the host controller aborts the queue advance and follows the queue head's horizontal pointer to the next schedule data structure. The host controller performs the overlay based on the following rules:

- The value of the data toggle (*dt*) field in the overlay area depends on the value of the *data toggle control (dtc)* bit (see Table 161 on page 405).



- If the *EPS* field indicates the endpoint is a high-speed endpoint, the *Ping* state field is preserved by the host controller. The value of this field is not changed as a result of the overlay.
- *C-prog-mask* field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- *Frame Tag* field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- *NakCnt* field in the overlay area is loaded from the *RL* field in the queue head's Static Endpoint State.
- All other areas of the overlay are set by the incoming qTD.

The host controller exits this state when it has committed the write to the queue head.

9.14.10.3 Execute Transaction

The host controller enters this state from the **Fetch Queue Head** state only if the *Active* bit in *Status* field of the queue head is set to a one.

On entry to this state, the host controller executes a few pre-operations, then checks some pre-condition criteria before committing to executing a transaction for the queue head.

The pre-operations performed and the pre-condition criteria depend on whether the queue head is an interrupt endpoint. The host controller can determine that a queue head is an interrupt queue head when the queue head's *S-mask* field contains a non-zero value. It is the responsibility of software to ensure the *S-mask* field is appropriately initialized based on the transfer type. There are other criteria that must be met if the *EPS* field indicates that the endpoint is a low- or full-speed endpoint, see [Section 9.14.12.1, "Split Transactions for Asynchronous Transfers" on page 453](#) and [Section 9.14.12.2, "Split Transaction Interrupt" on page 455](#).

- **Interrupt Transfer Pre-condition Criteria**
If the queue head is for an interrupt endpoint (e.g. non-zero *S-mask* field), then the *FRINDEX*[2:0] field must identify a bit in the *S-mask* field that has a one in it. For example, an *S-mask* value of 00100000b would evaluate to true only when *FRINDEX*[2:0] is equal to 101b. If this condition is met then the host controller considers this queue head for a transaction.
- **Asynchronous Transfer Pre-operations and Pre-condition Criteria**
If the queue head is not for an interrupt endpoint (e.g. a zero *S-mask* field), then the host controller performs one pre-operation and then evaluates one pre-condition criteria: The pre-operation is:
 - Checks the Nak counter reload state ([Section 9.14.9, "Operational Model for Nak Counter" on page 439](#)). It may be necessary for the host controller to reload the Nak Counter field. The reload is performed at this time.The pre-condition evaluated is:
 - Whether or not the *NakCnt* field has been reloaded, the host controller checks the value of the *NakCnt* field in the queue head. If *NakCnt* is non-zero, or if the *Reload Nak Counter* field is zero, then the host controller considers this queue head for a transaction.
- **Transfer Type Independent Pre-operations**
Regardless of the transfer type, the host controller always performs at least one pre-operation and evaluates one pre-condition. The pre-operation is:
 - A host controller internal transaction (down) counter *qHTransactionCounter* is loaded from the queue head's *Mult* field. A host controller implementation is allowed to ignore this for queue heads on the asynchronous list. It is



mandatory for interrupt queue heads. Software should ensure that the *Mult* field is set appropriately for the transfer type.

The pre-conditions evaluated are:

- The host controller determines whether there is enough time in the micro-frame to complete this transaction (see [Section 9.14.4.1, “Example: Preserving Micro-Frame Integrity” on page 420](#) for an example evaluation method). If there is not enough time to complete the transaction, the host controller exits this state.
- If the value of *qHTransactionCounter* for an interrupt endpoint is zero, then the host controller exits this state.

When the pre-operations are complete and pre-conditions are met, the host controller sets the *Reclamation* bit in the USBSTS register to a one and then begins executing one or more transactions using the endpoint information in the queue head. The host controller iterates *qHTransactionCounter* times in this state executing transactions. After each transaction is executed, *qHTransactionCounter* is decremented by one. The host controller will exit this state when one of the following events occurs:

- The *qHTransactionCounter* decrements to zero, or
- The endpoint responds to the transaction with any handshake other than an ACK,¹ or
- The transaction experiences a transaction error, or
- The *Active* bit in the queue head goes to a zero, or
- There is not enough time in the micro-frame left to execute the next transaction (see [Section 9.14.4.1.1, “Transaction Fit: A Best-Fit Approximation Algorithm” on page 420](#) for example method for implementing the frame boundary test).

The results of each transaction is recorded in the on-chip overlay area. If data was successfully moved during the transaction, the transfer state in the overlay area is advanced. To advance queue head's transfer state, the Total Bytes to Transfer field is decremented by the number of bytes moved in the transaction, the data toggle bit (*dt*) is toggled, the current page offset is advanced to the next appropriate value (e.g. advanced by the number of bytes successfully moved), and the *C_Page* field is updated to the appropriate value (if necessary). See [“Buffer Pointer List Use for Data Streaming with qTDs” on page 449](#).

Note that the Total Bytes To Transfer field may be zero when all the other criteria for executing a transaction are met. When this occurs, the host controller will execute a zero-length transaction to the endpoint. If the *PID_Code* field indicates an IN transaction and the device delivers data, the host controller will detect a packet babble condition, set the babble and halted bits in the Status field, set the *Active* bit to a zero, write back the results to the source qTD, then exit this state.

In the event an IN token receives a data PID mismatch response, the host controller must ignore the received data (e.g. not advance the transfer state for the bytes received). Additionally, if the endpoint is an interrupt IN, then the host controller must record that the transaction occurred (e.g. decrement *qHTransactionCounter*). It is recommended (but not required) the host controller continue executing transactions for this endpoint if the resultant value of *qHTransactionCounter* is greater than one.

If the response to the IN bus transaction is a Nak (or Nyet) and *RL* is non-zero, *NakCnt* is decremented by one. If *RL* is zero, then no write-back by the host controller is required (for a transaction receiving a Nak or Nyet response and the value of *CErr* did not change). Software should set the *RL* field to zero if the queue head is an interrupt endpoint. Host controller hardware is not required to enforce this rule or operation.

-
1. For a high-bandwidth interrupt OUT endpoint, the host controller may optionally immediately retry the transaction if it fails.



After the transaction has finished and the host controller has completed the post processing of the results (advancing the transfer state and possibly NakCnt, the host controller writes back the results of the transaction to the queue head's overlay area in main memory.

The number of bytes moved during an IN transaction depends on how much data the device endpoint delivers. The maximum number of bytes a device can send is Maximum Packet Size. The number of bytes moved during an OUT transaction is either Maximum Packet Length bytes or Total Bytes to Transfer, whichever is less.

If there was a transaction error during the transaction, the transfer state (as defined above) is not advanced by the host controller. The CErr field is decremented by one and the status field is updated to reflect the type of error observed. Transaction errors are summarized in [“Transaction Error” on page 483](#).

The following events will cause the host controller to clear the Active bit in the queue head's overlay status field. When the Active bit transitions from a one to a zero, the transfer in the overlay is considered complete. The reason for the transfer completion (clearing the Active bit) determines the next state.

- CErr field decrements to zero. When this occurs the *Halted* bit is set to a one and *Active* is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.
- The device responds to the transaction with a STALL PID. When this occurs, the *Halted* bit is set to a one and the *Active* bit is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.
- The *Total Bytes to Transfer* field is zero after the transaction completes. Note that for a zero length transaction, it was zero before the transaction was started. When this condition occurs, the *Active* bit is set to zero.
- The PID code is an IN, and the number of bytes moved during the transaction is less than the *Maximum Packet Length*. When this occurs, the *Active* bit is set to zero and a short packet condition exists. The short-packet condition is detected during the **Advance Queue** state. Refer to [Section 9.14.12, “Split Transactions” on page 453](#) for additional rules for managing low- and full-speed transactions.
- The *PID Code* field indicates an IN and the device sends more than the expected number of bytes (e.g. *Maximum Packet Length* or *Total Bytes to Transfer* bytes, whichever is less) (e.g. a packet babble). This results in the host controller setting the *Halted* bit to a one.

With the exception of a NAK response (when *RL* field is zero), the host controller always writes the results of the transaction back to the overlay area in main memory. This includes when the transfer completes. For a high-speed endpoint, the queue head information written back includes minimally the following fields:

- NakCnt, dt, Total Bytes to Transfer, C_Page, Status, CERR, and Current Offset

For a low- or full-speed device the queue head information written back also includes the fields:

- C-prog-mask, FrameTag and S-bytes.

The duration of this state depends on the time it takes to complete the transaction(s) and the status write to the overlay is committed.



9.14.10.3.1 Halting a Queue Head

A halted endpoint is defined only for the transfer types that are managed via queue heads (control, bulk and interrupt). The following events indicate that the endpoint has reached a condition where no more activity can occur without intervention from the driver:

- An endpoint may return a STALL handshake during a transaction,
- A transaction had three consecutive error conditions, or
- A Packet Babble error occurs on the endpoint.

When any of these events occur (for a queue head) the Host Controller halts the queue head and set the USBERRINT status bit in the USBSTS register to a one. To halt the queue head, the *Active* bit is set to a zero and the *Halted* bit is set to a one. There may be other error status bits that are set when a queue is halted. The host controller always writes back the overlay area to the source qTD when the transfer is complete, regardless of the reason (normal completion, short packet or halt). The host controller will not advance the transfer state on a transaction that results in a *Halt* condition (e.g. no updates necessary for *Total Bytes to Transfer*, *C_Page*, *Current Offset*, and *dt*). The host controller must update *CErr* as appropriate. When a queue head is halted, the *USB Error Interrupt* bit in the USBSTS register is set to a one. If the *USB Error Interrupt Enable* bit in the USBINTR register is set to a one, a hardware interrupt is generated at the next interrupt threshold.

9.14.10.3.2 Asynchronous Schedule Park Mode

Asynchronous Schedule Park mode is a special execution mode that can be enabled by system software, where the host controller is permitted to execute more than one bus transaction from a high-speed queue head in the Asynchronous schedule before continuing horizontal traversal of the Asynchronous schedule. This feature has no effect on queue heads or other data structures in the Periodic schedule. This feature is similar in intent as the *Mult* feature that is used in the Periodic schedule. Where-as the *Mult* feature is a characteristic that is tunable for each endpoint; park-mode is a policy that is applied to all high-speed queue heads in the asynchronous schedule. It is essentially the specification of an iterator for consecutive bus transactions to the same endpoint. All of the rules for managing bus transactions and the results of those as defined in [Section 9.14.10.3, "Execute Transaction" on page 444](#) apply. This feature merely specifies how many consecutive times the host controller is permitted to execute from the same queue head before moving to the next queue head in the Asynchronous List. This feature should allow the host controller to attain better bus utilization for those devices that are capable of moving data at maximum rate, while at the same time providing a fair service to all endpoints.

A host controller exports its capability to support this feature to system software by setting the *Asynchronous Schedule Park Capability* bit in the HCCPARAMS register to a one. This information keys system software that the *Asynchronous Schedule Park Mode Enable* and *Asynchronous Schedule Park Mode Count* fields in the USBCMD register are modifiable. System software enables the feature by writing a one to the *Asynchronous Schedule Park Mode Enable* bit.

When park-mode is not enabled (e.g. *Asynchronous Schedule Park Mode Enable* bit in the USBCMD register is a zero), the host controller must not execute more than one bus transaction per high-speed queue head, per traversal of the asynchronous schedule. When park-mode is enabled, the host controller must not apply the feature to a queue head whose *EPS* field indicates a Low/Full-speed device (i.e. only one bus transaction is allowed from each Low/Full-speed queue head per traversal of the asynchronous schedule). Park-mode may only be applied to queue heads in the Asynchronous schedule whose *EPS* field indicates that it is a high-speed device.



The host controller must apply park mode to queue heads whose *EPS* field indicates a high-speed endpoint. The maximum number of consecutive bus transactions a host controller may execute on a high-speed queue head is determined by the value in the *Asynchronous Schedule Park Mode Count* field in the USBCMD register. Software must not set *Asynchronous Schedule Park Mode Enable* bit to a one and also set *Asynchronous Schedule Park Mode Count* field to a zero. The resulting behavior is not defined. An example behavioral example describes the operational requirements for the host controller implementing park-mode. This feature does not affect how the host controller handles the bus transaction as defined in [Section 9.14.10.3, “Execute Transaction” on page 444](#). It only effects how many consecutive bus transactions for the current queue head can be executed. All boundary conditions, error detection and reporting applies as usual. This feature is similar in concept to the use of the *Mult* field for high-bandwidth Interrupt for queue heads in the Periodic Schedule.

The host controller effectively loads an internal down-counter *PM-Count* from *Asynchronous Schedule Park Mode Count* when *Asynchronous Schedule Park Mode Enable* bit is a one, and a high-speed queue head is first fetched and meets all the criteria for executing a bus transaction. After the bus transaction, *PM-Count* is decremented. The host controller may continue to execute bus transactions from the current queue head until *PM-Count* goes to zero, an error is detected, the buffer for the current transfer is exhausted or the endpoint responds with a flow-control or STALL handshake. [Table 176](#) summarizes the responses that effect whether the host controller continues with another bus transaction for the current queue head.

Table 176. Actions for Park Mode, Based on Endpoint Response and Residual Transfer State (Sheet 1 of 2)

PID	Endpoint Response	Transfer State after Transaction		Action
		PM-Count	Bytes to Transfer	
IN	DATA[0,1] w/ Maximum Packet sized data	Not zero	Not Zero	Allowed to perform another bus transaction. ^{1, 2}
		Not zero	Zero	Retire qTD and move to next QH
		Zero	Don't care	Move to next QH.
	DATA[0,1] w/ short packet	Don't care	Don't care	Retire qTD and move to next QH.
	NAK	Don't care	Don't care	Move to next QH.
	STALL, XactErr	Don't care	Don't care	Move to next QH.
OUT	ACK	Not zero	Not Zero	Allowed to perform another bus transaction. ²
		Not zero	Zero	Retire qTD and move to next QH
		Zero	Don't care	Move to next QH.
	NYET, NAK	Don't care	Don't care	Move to next QH.
	STALL, XactErr	Don't care	Don't care	Move to next QH
Notes: 1. The host controller may continue to execute bus transactions from the current high-speed queue head (if PM-Count is not equal to zero), if a PID mismatch is detected (e.g. expected DATA1 and received DATA0, or visa-versa). 2. This specification does not require that the host controller execute another bus transaction when PM-Count is non-zero. Implementations are encouraged to make appropriate complexity and performance trade-offs.				

**Table 176. Actions for Park Mode, Based on Endpoint Response and Residual Transfer State (Sheet 2 of 2)**

PID	Endpoint Response	Transfer State after Transaction		Action
		PM-Count	Bytes to Transfer	
PING	ACK	Not Zero	Not Zero	Allowed to perform another bus transaction. ²
	NAK	Don't care	Don't care	Move to next QH
	STALL, XactErr	Don't care	Don't care	Move to next QH
Notes: 1. The host controller may continue to execute bus transactions from the current high-speed queue head (if PM-Count is not equal to zero), if a PID mismatch is detected (e.g. expected DATA1 and received DATA0, or visa-versa). 2. This specification does not require that the host controller execute another bus transaction when PM-Count is non-zero. Implementations are encouraged to make appropriate complexity and performance trade-offs.				

9.14.10.4 Write Back qTD

This state is entered from the **Execute Transaction** state when the *Active* bit is set to a zero. The source data for the write-back is the transfer results area of the queue head overlay area (see [Figure 50, "Queue Head Structure Layout" on page 404](#)). The host controller uses the *Current qTD Pointer* field as the target address for the qTD. The queue head transfer result area is written back to the transfer result area of the target qTD. This state is also referred to as: qTD retirement. The fields that must be written back to the source qTD include *Total Bytes to Transfer*, *Cerr*, and *Status*.

The duration of this state depends on when the qTD write-back has been committed.

9.14.10.5 Follow Queue Head Horizontal Pointer

The host controller must use the horizontal pointer in the queue head to the next schedule data structure when any of the following conditions exist:

- If the *Active* bit is a one on exit from the **Execute Transaction** state, or
- When the host controller exits the **Write Back qTD** state, or
- If the **Advance Queue** state fails to advance the queue because the target qTD is not active, or
- If the *Halted* bit is a one on exit from the **Fetch QH** state.

There is no functional requirement that the host controller wait until the current transaction is complete before using the horizontal pointer to read the next linked data structure. However, it must wait until the current transaction is complete before executing the next data structure.

9.14.10.6 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers, which is used to reference the data buffer for a transfer. This specification requires that the buffer associated with the transfer be *virtually contiguous*. This means: if the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to



a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

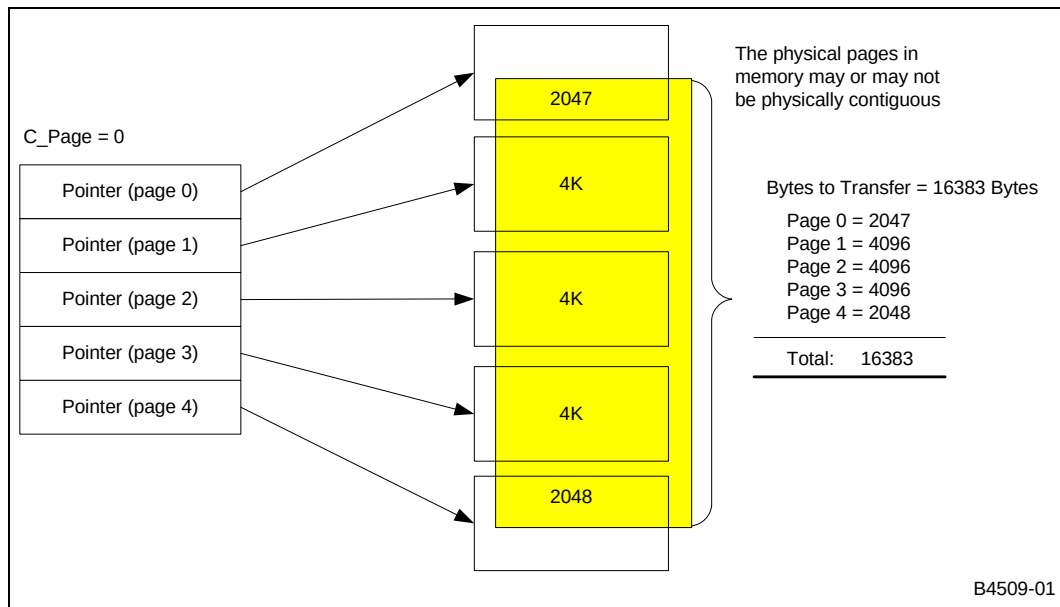
The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16-Kbyte buffer with any starting buffer alignment.

The host controller uses the field *C_Page* field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction will span a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing *C_Page* and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the *Bytes to Transfer* field.

Figure 66, “Example Mapping of qTD Buffer Pointers to Buffer Pages” on page 450 illustrates a nominal example of how System software would initialize the buffer pointers list and the *C_Page* field for a transfer size of 16,383 bytes. *C_Page* is set to zero. The upper 20-bits of Page 0 references the start of the physical page. *Current Offset* (the lower 12-bits of queue head Dword 7) holds the offset in the page e.g. 2,049 (e.g. 4,096-2,047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

Figure 66. Example Mapping of qTD Buffer Pointers to Buffer Pages



For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because *C_Page* is set to zero) and concatenates the *Current Offset* field. The 512 bytes are moved during the transaction, the *Current Offset* and *Total Bytes to Transfer* are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller will increment *C_Page* (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page



pointer is the page 1 pointer and *Current Offset* has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (i.e. *C_Page*) when necessary. There are three conditions for how the host controller handles *C_Page*.

- The current transaction does not span a page boundary. The value of *C_Page* is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (i.e. the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment *C_Page* before writing back status for the transaction.

Note that the only valid adjustment the host controller may make to *C_Page* is to increment by one.

9.14.10.7 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's *S-Mask* to indicate which micro-frame with-in a 1-ms period a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have *S-Mask* set to a non-zero value. An *S-mask* with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and *S-Mask* values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in [Table 177](#).

Table 177. Example Periodic Reference Patterns for Interrupt Transfers with 2-ms Poll Rate

Frame # Reference Sequence	Description
0, 2, 4, 6, 8, etc. <i>S-Mask</i> = 01h	A queue head for the <i>bInterval</i> of 2 ms (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the <i>S-Mask</i> field in the queue head is set to 01h, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8, etc. <i>S-Mask</i> = 02h	Another example of a queue head with a <i>bInterval</i> of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the <i>S-Mask</i> is set to 02h indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

9.14.10.8 Managing Transfer Complete Interrupts from Queue Heads

The host controller will set an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an *Interrupt on Complete (IOC)* bit set to a one, or whenever a transfer (qTD) completes with a short packet. If system software needs multiple qTDs to complete a client request (i.e. like a control transfer) the intermediate qTDs do not require interrupts. System software may only need a single interrupt to notify it that the complete buffer has been transferred. System software may set *IOC*'s to occur more frequently. A motivation for this may be that it wants early notification so that interface data structures can be re-used in a timely manner.



9.14.11 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called Ping. Ping is required for all USB 2.0 High-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The *Status* field has a *Ping State* bit, which the host controller uses to determine the *next* actual PID it will use in the next transaction to the endpoint (see Table 158, “qTD Token (DWord 2)” on page 401). The Ping State bit is only managed by the host controller for queue heads that meet the following criteria:

- Queue head is not an interrupt and
- *EPS* field equals High-Speed and
- *PIDCode* field equals OUT

Table 178, “Ping Control State Transition Table” on page 452 illustrates the state transition table for the host controller’s responsibility for maintaining the PING protocol. Refer to Chapter 8 in the USB Specification Revision 2.0 for detailed description on the Ping protocol.

Table 178. Ping Control State Transition Table

Current	Event		Next
	Host	Device	
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr ¹	Do Ping
Do Ping	PING	Stall	N/C ²
OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping ¹
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr ¹	Do Ping
Do OUT	OUT	Stall	N/C ²
Notes: 1. Transaction Error (XactErr) is any time the host misses the handshake. 2. No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (e.g. Active set to zero and Halt set to a one). Software intervention is required to restart queue. 3. A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.			

The Ping State bit has the following encoding:

Table 179. Ping State Encoding

Value	Meaning
0B	Do OUT The host controller will use an OUT PID during the next bus transaction to this endpoint.
1B	Do Ping The host controller will use a PING PID during the next bus transaction to this endpoint.

The defined ping protocol (see USB 2.0 Specification, Chapter 8) allows the host to be *imprecise* on the initialization of the ping protocol (i.e. start in *Do OUT* when we don’t know whether there is space on the device or not). The host controller manages the *Ping State* bit. System software sets the initial value in the queue head when it



initializes a queue head. The host controller preserves the *Ping State* bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the *Ping State* bit is preserved.

9.14.12 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB-1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below USB 2.0 hub, utilizing the split transaction protocol. Refer to USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and Low-speed devices are enumerated identically as high-speed devices, but the transactions to the Full- and Low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the Full- or Low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and Transaction Translator below which the Full- or Low-speed device is attached.

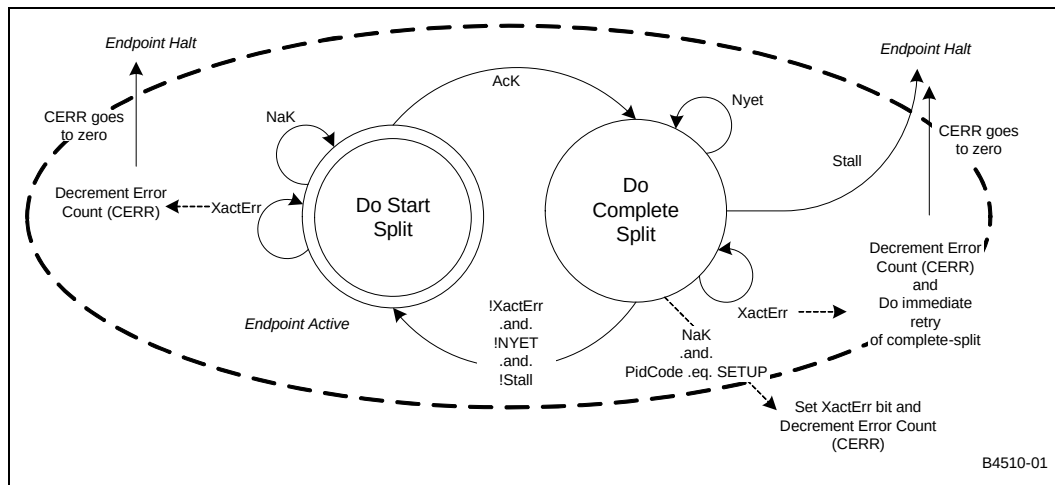
The EHCI interface uses dedicated data structures for managing full-speed isochronous data streams (see [Section 9.13.4, “Split Transaction Isochronous Transfer Descriptor \(siTD\)” on page 395](#)). Control, Bulk and Interrupt are managed using the queuing data structures. The interface data structures need to be programmed with the device address and the Transaction Translator number of the USB 2.0 hub operating as the Low-/Full-speed host controller for this link. The following sections describe the details of how the host controller must process and manage the split transaction protocol.

9.14.12.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an *EPS* field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full-, low-speed control are managed via queue heads in the asynchronous schedule.

Software must initialize the queue head with the appropriate device address and port number for the transaction translator that is serving as the full/low-speed host controller for the links connecting the endpoint. Software must also initialize the split transaction state bit (*SplitXState*) to **Do-Start-Split**. Finally, if the endpoint is a control endpoint, then system software must set the *Control Transfer Type* (*C*) bit in the queue head to a one. If this is not a control transfer type endpoint, the *C* bit must be initialized by software to be a zero. This information is used by the host controller to properly set the Endpoint Type (*ET*) field in the split transaction bus token. When the *C* bit is a zero, the split transaction token's *ET* field is set to indicate a bulk endpoint. When the *C* bit is a one, the split transaction token's *ET* field is set to indicate a control endpoint. Refer to Chapter 8 of USB Specification Revision 2.0 for details.

Figure 67. Host Controller Asynchronous Schedule Split-Transaction State Machine



9.14.12.1.1 Asynchronous — Do Start Split

This is the state which software must initialize a full- or low-speed asynchronous queue head. This state is entered from the **Do Complete Split** state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller will execute a start-split transaction to the appropriate transaction translator. If the bus transaction completes without an error and *PidCode* indicates an IN or OUT transaction, then the host controller will reload the error counter (*CErr*). If it is a successful bus transaction and the *PidCode* indicates a SETUP, the host controller will not reload the error counter. If the transaction translator responds with a Nak, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response) the host controller decrements $Cerr$ and proceeds to the next queue head in the asynchronous schedule.

9.14.12.1.2 Asynchronous — Do Complete Split

This state is entered from the **Do Start Split** state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller will execute a complete-split transaction to the appropriate transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's *PidCode* indicates an IN or OUT, the host controller will reload the error counter (*CErr*). When a Nyet handshake is received for a complete-split bus transaction where the queue head's *PidCode* indicates a SETUP, the host controller must not adjust the value of *CErr*.

Independent of *PIDCode*, the following responses have the effects:

- Transaction Error (XactErr). Timeout or data CRC failure, etc. The error counter (Cerr) is decremented by one and the complete split transaction is *immediately* retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller **MUST** ensure that the next time the host controller begins



executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. A method to accomplish this behavior is to not advance the asynchronous schedule. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule will be the retry for this endpoint. If *Cerr* went to zero, the host controller must halt the queue.

- **NAK.** The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the *PidCode* is a SETUP, then the Nak response is a protocol error. The *XactErr* status bit is set to a one and the *CErr* field is decremented.
- **STALL.** The target endpoint responded with a STALL handshake. The host controller sets the *halt* bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the *PidCode* indicates an IN, then any of following responses are expected:

- **DATA0/1.** On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is *good*, the host controller will advance the state of the transfer, e.g. move the data pointer by the number of bytes received, decrement *BytesToTransfer* field by the number of bytes received, and toggle the *dt* bit. The host controller will then exit this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited. If the *PidCode* indicates an OUT/SETUP, then any of following responses are expected:

- **ACK.** The target endpoint accepted the data, so the host controller must advance the state of the transfer. The *Current Offset* field is incremented by *Maximum Packet Length* or *Bytes to Transfer*, whichever is less. The field *Bytes To Transfer* is decremented by the same amount and the data toggle bit (*dt*) is toggled. The host controller will then exit this state.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue (see [Section 9.14.10, "Managing Control/Bulk/Interrupt Transfers via Queue Heads" on page 441](#)).

9.14.12.2 Split Transaction Interrupt

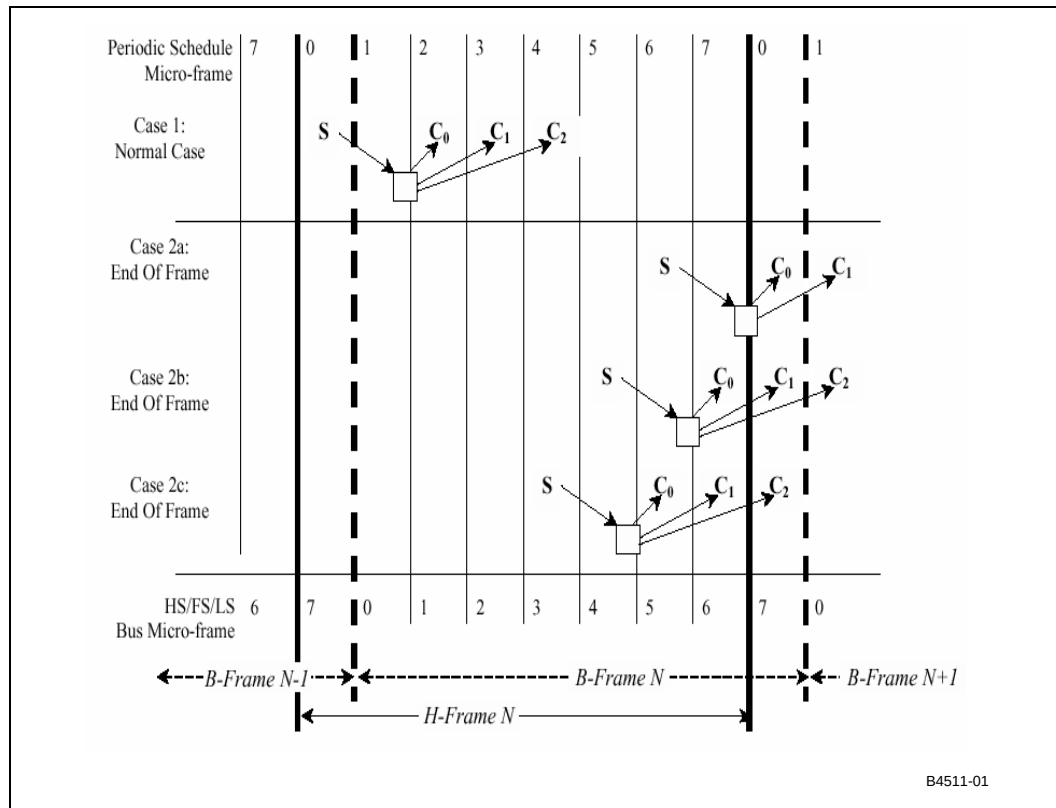
Split-transaction Interrupt-IN/OUT endpoints are managed via the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, for a high-speed endpoint, the host controller will visit a queue head, execute a high-speed transaction (if criteria are met) and advance the transfer state (or not) depending on the results of the entire transaction. For low- and full-speed endpoints, the details of the *execution* phase are different (i.e. takes more than one bus transaction to complete), but the remainder of the operational framework is intact. This means that the transfer advancement, etc. occurs as defined in [Section 9.14.10, "Managing Control/Bulk/Interrupt Transfers via Queue Heads" on page 441](#), but only occurs on the completion of a split transaction.

9.14.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an *EPS* field indicating full- or low-speed and have a non-zero *S-mask* field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 68 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The **S** and **C_X** labels indicate micro-frames where software can schedule start-splits and complete splits, respectively.

Figure 68. Split Transaction, Interrupt Scheduling Boundary Conditions



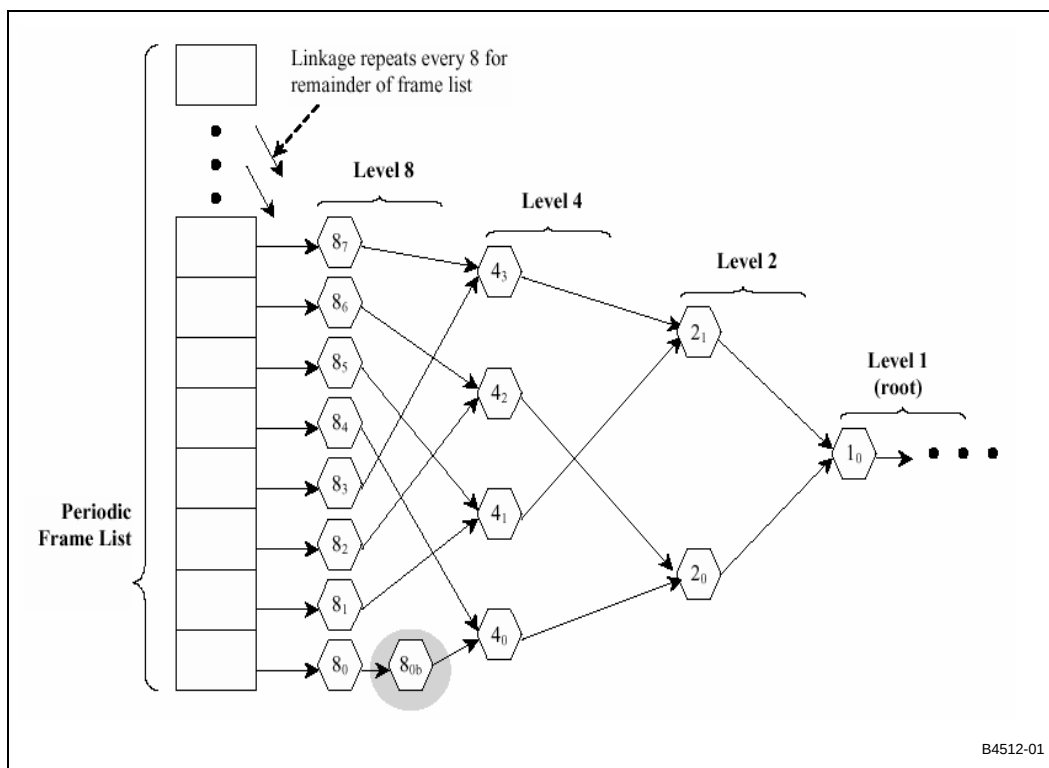
The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (*H-Frame* in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the *H-Frame* boundary when the start-split is in micro-frame 4 or later. When this occurs, the *H-*



Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. Figure 69, “General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading” on page 457 illustrates the general layout of the periodic schedule.

Figure 69. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading



The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a 2^N poll rate. Software can efficiently manage periodic bandwidth on the USB by *spreading* interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate eight queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 8_{0b} where such an endpoint. Without additional support on the interface, to get 8_{0b} reachable at the correct time, software would have to link 8₁ to 8_{0b}. It would then have to move 4₁ and everything linked after into the same path as 4₀. This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. “Host Controller Operational Model for FSTNs” on page 458 defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by system software to instruct the host controller when to execute portions of the split-transaction protocol.



- *SplitXState*. This is a single bit residing in the *Status* field of a queue head (see [Table 158, “qTD Token \(DWord 2\)” on page 401](#)). This bit is used to track the current state of the split transaction.
- *Frame S-mask*. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an *H-Frame*) that the host controller should execute a start-split transaction. This is always qualified by the value of the *SplitXState* bit in the *Status* field of the queue head. For example, referring to [Table 68, “Split Transaction, Interrupt Scheduling Boundary Conditions” on page 456](#), case one, the *S-mask* would have a value of 00000001b indicating that if the queue head is traversed by the host controller, and the *SplitXState* indicates **Do_Start**, and the current micro-frame as indicated by *FRINDEX[2:0]* is 0, then execute a start-split transaction.
- *Frame C-mask*. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an *H-Frame*) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the *SplitXState* bit in the *Status* field of the queue head. For example, referring to [Table 68, “Split Transaction, Interrupt Scheduling Boundary Conditions” on page 456](#), case one, the *C-mask* would have a value of 00011100b indicating that if the queue head is traversed by the host controller, and the *SplitXState* indicates **Do_Complete**, and the current micro-frame as indicated by *FRINDEX[2:0]* is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between *H-Frames* and *B-Frames* is correctly performed when setting bits in *S-mask* and *C-mask*.

9.14.12.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure is used to manage Low/Full-speed interrupt queue heads that need to be reached from consecutive frame list locations (i.e. boundary cases 2a through 2c). An FSTN is essentially a *back pointer*, similar in intent to the back pointer field in the siTD data structure (see [Section 9.13.4.5, “siTD Back Link Pointer” on page 398](#)).

This feature provides software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in [“Periodic Frame Span Traversal Node \(FSTN\)” on page 408](#).
- A *Save Place indicator*. This is always an FSTN with its *Back Path Link Pointer.T-bit* set to zero.
- A *Restore indicator*. This is always an FSTN with its *Back Path Link Pointer.T-bit* set to a one.
- Host controller FSTN traversal rules.

Host Controller Operational Model for FSTNs

When the host controller encounters an FSTN during micro-frames 2 through 7 it simply follows the node's *Normal Path Link Pointer* to access the next schedule data structure. Note that the FSTN's *Normal Path Link Pointer.T-bit* may set to a one, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a *Save-Place* FSTN in micro-frames 0 or 1, it will save the value of the *Normal Path Link Pointer* and set an internal flag indicating that it is executing in *Recovery Path* mode. *Recovery Path* mode modifies the host controller's rules for how it traverses the schedule and limits which data structures will be



considered for execution of bus transactions. The host controller continues executing in *Recovery Path* mode until it encounters a *Restore* FSTN or it determines that it has reached the end of the micro-frame (see details in the list below).

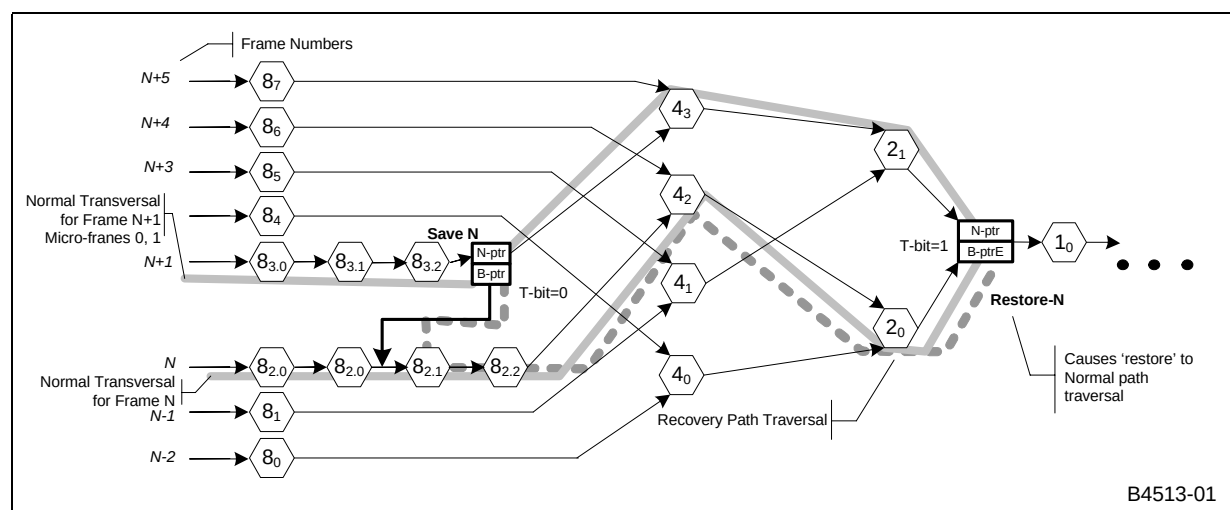
The rules for schedule traversal and limited execution while in *Recovery Path* mode are:

- Always follow the *Normal Path Link Pointer* when it encounters an FSTN that is a *Save-Place* indicator. The host controller must not recursively follow *Save-Place* FSTNs. Therefore, while executing in *Recovery Path* mode, it must never follow an FSTN's *Back Path Link Pointer*.
- Do not process an siTD or, iTD data structure. Simply follow its *Next Link Pointer*.
- Do not process a QH (Queue Head) whose *EPS* field indicates a high-speed device. Simply follow its *Horizontal Link Pointer*.
- When a QH's *EPS* field indicates a Full/Low-speed device, the host controller will only consider it for execution if its *SplitXState* is **DoComplete** (note: this applies whether the *PID Code* indicates an IN or an OUT). See [Section 9.14.10.3, "Execute Transaction" on page 444](#) and ["Tracking Split Transaction Progress for Interrupt Transfers" on page 461](#) for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in *Recovery Path* mode. See ["Periodic Interrupt - Do Complete Split" on page 464](#) for special handling when in *Recovery Path* mode.
- Stop traversing the *recovery path* when it encounters an FSTN that is a *Restore* indicator. The host controller unconditionally uses the saved value of the *Save-Place* FSTN's *Normal Path Link Pointer* when returning to the normal path traversal. The host controller must clear the context of executing a *Recovery Path* when it restores schedule traversal to the *Save-Place* FSTN's *Normal Path Link Pointer*.

If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in [Figure 70](#).

Figure 70. Example Host Controller Traversal of Recovery Path via FSTNs





In frame N (micro-frames 0-7), for this example, the host controller will traverse all of the schedule data structures utilizing the *Normal Path Link Pointers* in any FSTNs it encounters. This is because the host controller has not yet encountered a *Save-Place* FSTN so it not executing in *Recovery Path* mode. When it encounters the *Restore* FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N.Normal Path Link Pointer to traverse to the next data structure (i.e. normal schedule traversal). This is because the host controller must use a Restore FSTN's *Normal Path Link Pointer* when not executing in a *Recovery-Path* mode. The nodes traversed during frame N include: {8_{2.0}, 8_{2.1}, 8_{2.2}, 8_{2.3}, 4₂, 2₀, Restore-N, 1₀ ...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters *Save-Path* FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a *Save-Path* indicator). The host controller saves the value of Save-N.Normal Path Link Pointer and follows Save-N.Back Path Link Pointer. At the same time, it sets an internal flag indicating that it is now in *Recovery Path* mode (the recovery path is annotated in Figure 70, "Example Host Controller Traversal of Recovery Path via FSTNs" on page 459 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches *Restore* FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set to a one (definition of a *Restore* indicator), so the host controller exits *Recovery Path* mode by clearing the internal *Recovery Path* mode flag and commences (restores) schedule traversal using the saved value of the *Save-Place* FSTN's *Normal Path Link Pointer* (e.g. Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {8_{3.0}, 8_{3.1}, 8_{3.2}, Save-A, **8_{2.2}, 8_{2.3}, 4₂, 2₀, Restore-N**, 4₃, 2₁, Restore-N, 1₀ ...}. The nodes on the recovery-path are bold faced.

In frame N+1 (micro-frames 2-7), when the host controller encounters *Save-Path* FSTN Save-N, it will unconditionally follow Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {8_{3.0}, 8_{3.1}, 8_{3.2}, Save-A, 4₃, 2₁, Restore-N, 1₀ ...}.

Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each *Save-Place* indicator requires a matching *Restore* indicator.

The *Save-Place* indicator is an FSTN with a valid *Back Path Link Pointer* and *T-bit* equal to zero. Note that *Back Path Link Pointer.Type* field must be set to indicate the referenced data structure is a queue head. The *Restore* indicator is an FSTN with its *Back Path Link Pointer.T-bit* set to a one.

A *Restore* FSTN may be matched to one or more *Save-Place* FSTNs. For example, if the schedule includes a poll-rate 1 level, then system software only needs to place a *Restore* FSTN at the beginning of this list in order to match all possible *Save-Place* FSTNs.

- If the schedule does not have elements linked at a poll-rate level of one, and one or more *Save-Place* FSTNs are used, then System Software must ensure the *Restore* FSTN's *Normal Path Link Pointer's T-bit* is set to a one, as this will be use to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a *Restore* FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that *Recovery Path* mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A *Save-Place* FSTN's *Back Path Link Pointer* must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the *Save-Place* FSTN is reachable from frame list



offset N, then the FSTN's *Back Path Link Pointer* must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one *Save-Place* FSTN reachable in any single frame. Note there will be times when two (or more, depending on the implementation) could exist as full/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the *Save-Place* FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

9.14.12.2.3 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue must be halted, thus signaling system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- *C-prog-mask*. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. *C-prog-mask* is a simple bit-vector that the host controller sets one of the *C-prog-mask* bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks *C-prog-mask* before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- *FrameTag*. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (*H-Frame* number) when the next complete split must be executed.
- *S-bytes*. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The *S-bytes* field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

9.14.12.2.4 Split Transaction Execution State Machine for Interrupt

In the following presentation, all references to micro-frame are in the context of a micro-frame within an *H-Frame*.



As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- *cMicroFrameBit*. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the *FRINDEX* register (i.e. $cMicroFrameBit = (1 \text{ shifted-left}(FRINDEX[2:0]))$). The *cMicroFrameBit* has at most one bit asserted, which always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then *cMicroFrameBit* will equal 00000001b.

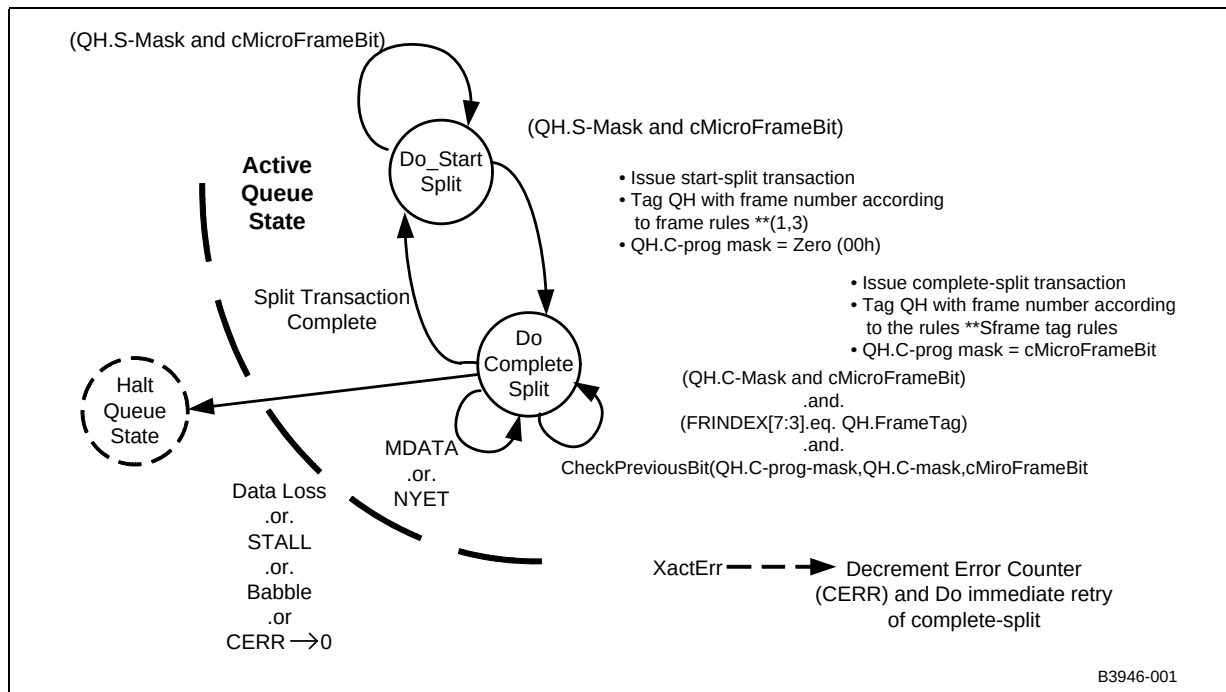
The variable *cMicroFrameBit* is used to compare against the *S-mask* and *C-mask* fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 71 illustrates the state machine for managing a complete interrupt split transaction. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the *SplitXState* is at **Do_Start** and the single bit in *cMicroFrameBit* has a corresponding bit active in *QH.S-mask*. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to **Do_Complete**. Due to the available jitter in the transaction translator pipeline, there will be more than one complete-split transaction scheduled by software for the **Do_Complete** state. This translates simply to the fact that there are multiple bits set to a one in the *QH.C-mask* field.

The host controller keeps the queue head in the **Do_Complete** state until the split transaction is complete (see definition below), or an error condition triggers the *three-strikes-rule* (e.g. after the host tries the same transaction three times, and each encounters an error, the host controller will stop retrying the bus transaction and halt the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).



Figure 71. Split Transaction State Machine for Interrupt



See “Managing QH.FrameTag Field” on page 467 for the frame tag management rules.

Periodic Interrupt - Do Start Split

This is the state software must initialize a full- or low-speed interrupt queue head *StartXState* bit. This state is entered from the **Do_Complete Split** state only after the split transaction is complete. This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (e.g. timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see “Periodic Interrupt - Do Complete Split” on page 464 for the definition of ‘Last’.

Each time the host controller visits a queue head in this state (once within the **Execute Transaction** state), it performs the following test to determine whether to execute a start-split.

- *QH.S-mask* is bit-wise *anded* with *cMicroFrameBit* (i.e. the and of the two bits).



If the result is non-zero, then the host controller will issue a start-split transaction. If the *PIDCode* field indicates an IN transaction, the host controller must zero-out the *QH.S-bytes* field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into *QH.FrameTag* field (see “[Managing QH.FrameTag Field](#)” on page 467), set *C-prog-mask* to zero (00h), and exits this state. Note that the host controller must not adjust the value of *CErr* as a result of completion of a start-split transaction.

Periodic Interrupt - Do Complete Split

This state is entered unconditionally from the **Do Start Split** state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the **Execute Transaction** state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- **Test A.** *cMicroFrameBit* is bit-wise *anded* with *QH.C-mask* field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame.
- **Test B.** *QH.FrameTag* is compared with the current contents of *FRINDEX[7:3]*. An equal indicates a match.
- **Test C.** The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

Algorithm Boolean CheckPreviousBit(*QH.C-prog-mask*, *QH.C-mask*, *cMicroFrameBit*)

Begin

-- Return values:

-- TRUE - no error

-- FALSE - error

--

Boolean rvalue = TRUE;

previousBit = *cMicroFrameBit* logical-rotate-right(1)

-- Bit-wise anding previousBit with C-mask indicates

-- whether there was an intent

-- to send a complete split in the previous micro-frame. So,

-- if the

-- 'previous bit' is set in C-mask, check C-prog-mask to

-- make sure it

-- happened.

If (previousBit **bitAND** *QH.C-mask*) then

 If **not**(previousBit **bitAND** *QH.C-prog-mask*) then

 rvalue = FALSE;



```

        End if

    End If

    -- If the C-prog-mask already has a one in this bit position,
    -- then an aliasing
    -- error has occurred. It will probably get caught by the
    -- FrameTag Test, but
    -- at any rate it is an error condition that as detectable here
    -- should not allow
    -- a transaction to be executed.

    If (cMicroFrameBit bitAND QH.C-prog-mask) then

        rvalue = FALSE;

    End if

    return (rvalue)

End Algorithm

```

- **Test D.** Check to see if a start-split should be executed in this micro-frame. Note this is the same test performed in the **Do Start Split** state (see [“Periodic Interrupt - Do Complete Split” on page 464](#)). Whenever it evaluates to TRUE and the controller is NOT processing in the context of a *Recovery Path* mode, it means a start-split should occur in this micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A and B and C and not (D)), the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates *QH.C-prog-mask* by bit-ORing with *cMicroFrameBit*. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets *QH.FrameTag* to the expected *H-Frame* number (see [“Managing QH.FrameTag Field” on page 467](#)). The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the *CErr* will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.

The test for whether this is the **Last** complete split can be performed by XOR *QH.C-mask* with *QH.C-prog-mask*. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the *XactErr* status bit is set to a one and the *CErr* field is decremented.

- NYET (and not Last). See above description for testing for **Last**. The complete-split transaction received a NYET response from the transaction translator. Do not



update any transfer state (except for *C-prog-mask* and *FrameTag*) and stay in this state. The host controller must not adjust *CErr* on this response.

- Transaction Error (*XactErr*). Timeout, data CRC failure, etc. The *CErr* field is decremented and the *XactErr* bit in the *Status* field is set to a one. The complete split transaction is *immediately* retried (if *Cerr* is non-zero). If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or *Cerr* is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and *Cerr* is not zero, then this state is exited (i.e. return to **Do Start Split**). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section full- and low-speed Interrupts) in the USB Specification Revision 2.0 for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The *Current Offset* field is incremented by *Maximum Packet Length* or *Bytes to Transfer*, whichever is less. The field *Bytes To Transfer* is decremented by the same amount. And the data toggle bit (*dt*) is toggled. The host controller will then exit this state for this queue head. The host controller must reload *CErr* with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue (see [Section 9.14.10, "Managing Control/Bulk/Interrupt Transfers via Queue Heads" on page 441](#)).
- MDATA. This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in *QH.S-bytes*. The host controller must not adjust *CErr* on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in *QH.S-bytes*. The state of the transfer is advanced by the result and the host controller will exit this state for this queue head.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue (see [Section 9.14.10, "Managing Control/Bulk/Interrupt Transfers via Queue Heads" on page 441](#)).

If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.

- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload *CErr* with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set to a one, *Cerr* is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the **Execute Transaction** state). The status field bits: *Active* bit is set to zero and the *Halted* bit is set to a one and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. [Table 180](#) lists the possible combinations and the appropriate action.

**Table 180. Interrupt IN/OUT Do Complete Split State Execution Criteria**

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current micro-frame. Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	Progress bit check failed. This means a complete-split has been missed. There is the possibility of lost data. If <i>PIDCode</i> is an IN, then the Queue head must be halted. If <i>PIDCode</i> is an OUT, then the transfer state is not advanced and the state exited (e.g. start-split is retried). This is a host-induced error and does not effect <i>CERR</i> . In either case, set the <i>Missed Micro-frame</i> bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	<i>QH.FrameTag</i> test failed. This means that exactly one or more <i>H-Frames</i> have been skipped. This means complete-splits have been missed. There is the possibility of lost data. If <i>PIDCode</i> is an IN, then the Queue head must be halted. If <i>PIDCode</i> is an OUT, then the transfer state is not advanced and the state exited (e.g. start-split is retried). This is a host-induced error and does not effect <i>CERR</i> . In either case, set the <i>Missed Micro-frame</i> bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If <i>PIDCode</i> is an IN, then the Queue head must be halted. If <i>PIDCode</i> is an OUT, then the transfer state is not advanced and the state exited (e.g. start-split is retried). This is a host-induced error and does not effect <i>CERR</i> . In either case, set the <i>Missed Micro-frame</i> bit in the status field to a one. Note: When executing in the context of a <i>Recovery Path</i> mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a <i>Recovery Path</i> mode.

Managing QH.FrameTag Field

The *QH.FrameTag* field in a queue head is completely managed by the host controller. The rules for setting *QH.FrameTag* are simple:

- **Rule 1:** If transitioning from **Do Start Split** to **Do Complete Split** and the current value of *FRINDEX*[2:0] is 6, *QH.FrameTag* is set to *FRINDEX*[7:3] + 1. This accommodates split transactions whose start-split and complete-splits are in different *H-Frames* (case 2a, see Table 68, “Split Transaction, Interrupt Scheduling Boundary Conditions” on page 456).
- **Rule 2:** If the current value of *FRINDEX*[2:0] is 7, *QH.FrameTag* is set to *FRINDEX*[7:3] + 1. This accommodates staying in **Do Complete Split** for cases 2a, 2b, and 2c (Table 68, “Split Transaction, Interrupt Scheduling Boundary Conditions” on page 456).
- **Rule 3:** If transitioning from **Do Start Split** to **Do Complete Split** and the current value of *FRINDEX*[2:0] is not 6, or currently in **Do Complete Split** and the current value of (*FRINDEX*[2:0]) is not 7, *FrameTag* is set to *FRINDEX*[7:3]. This accommodates all other cases (Table 68, “Split Transaction, Interrupt Scheduling Boundary Conditions” on page 456).

9.14.12.2.5 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (i.e. new S-mask and C-mask values).



It is imperative that System software must not update these masks to new values in the midst of a split transaction. In order to avoid any race conditions with the update, the EHCI host controller provides a simple assist to system software. System software sets the *Inactivate-on-next-Transaction (I)* bit to a one to signal the host controller that it intends to update the S-mask and C-mask on this queue head. System software will then wait for the host controller to observe the *I-bit* is a one and transition the *Active* bit to a zero. The rules for how and when the host controller sets the *Active* bit to zero are enumerated below:

- If the *Active* bit is a zero, no action is taken. The host controller does not attempt to advance the queue when the *I-bit* is a one.
- If the *Active* bit is a one and the *SplitXState* is **DoStart** (regardless of the value of *S-mask*), the host controller will simply set *Active* bit to a zero. The host controller is not required to write the transfer state back to the *current* qTD. Note that if the *S-mask* indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction. It must set the *Active* bit to zero.

System software must save transfer state before setting the *I-bit* to a one. This is required so that it can correctly determine what transfer progress (if any) occurred after the *I-bit* was set to a one and the host controller executed its final bus-transaction and set *Active* to a zero.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the *Active* bit and the *I-bit* cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped via the *I-bit*.

3. Set the *Halted* bit to a one, then
4. Set the *I-bit* to a zero, then
5. Set the *Active* bit to a one and the *Halted* bit to a zero in the same write.

Setting the *Halted* bit to a one inhibits the host controller from attempting to advance the queue between the time the *I-bit* goes to a zero and the *Active* bit goes to a one.

9.14.12.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The EHCI controller utilizes siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (iT_D, [Section 9.13.3, “Isochronous \(High-Speed\) Transfer Descriptor \(iT_D\)” on page 391](#)) (see [Section 9.14.7, “Managing Isochronous Transfers Using iT_Ds” on page 426](#) for the operational model of iT_Ds) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

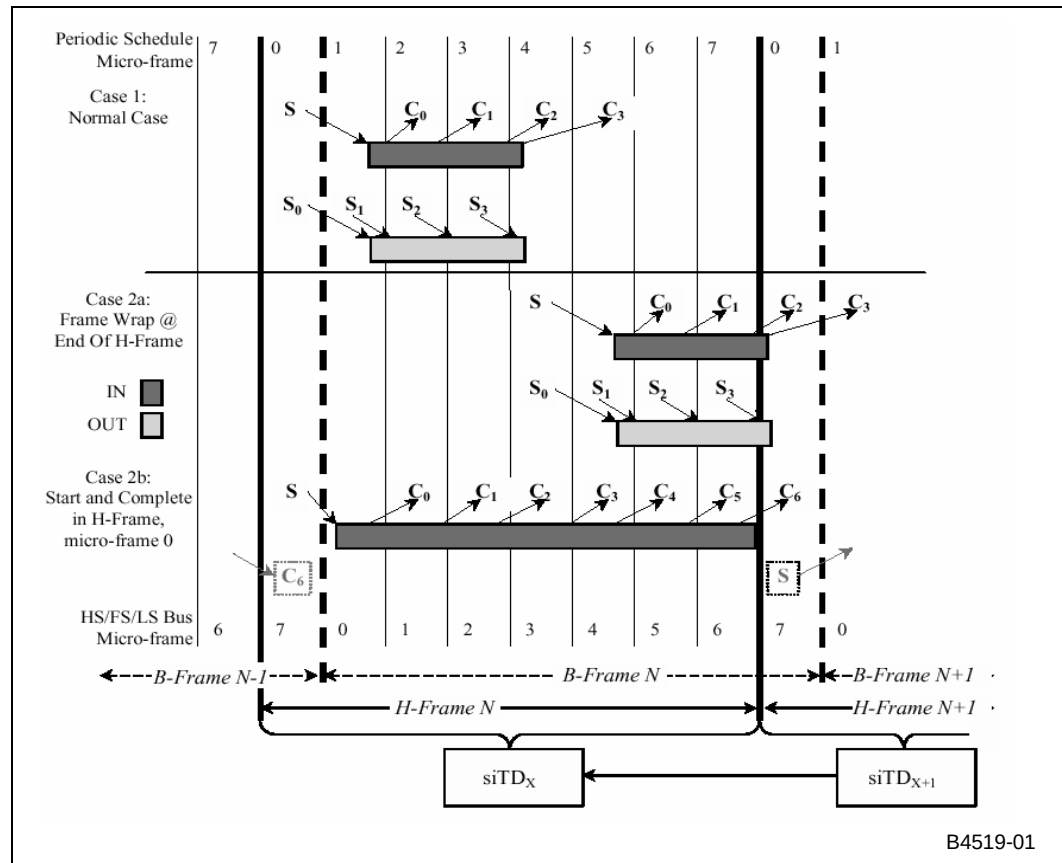
9.14.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in [“Split Transaction Scheduling Mechanisms for Interrupt” on page 456](#) apply. [Figure 72](#) illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The ^SX and ^CX labels indicate micro-frames where software can schedule start- and



complete-splits (respectively). The *H-Frame* boundaries are marked with a large, solid bold vertical line. The *B-Frame* boundaries are marked with a large, bold, dashed line. The bottom of the figure illustrates the relationship of an siTD to the *H-Frame*.

Figure 72. Split Transaction, Isochronous Scheduling Boundary Conditions



When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to *N* complete-splits. The scheduling boundary cases are:

- **Case 1:** The entire split transaction is completely bounded by an *H-Frame*. For example: the start-splits and complete-splits are all scheduled to occur in the same *H-Frame*.
- **Case 2a:** This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an *H-Frame* boundary. This can only occur when the split transaction has the possibility of moving data in *B-Frame*, micro-frames 6 or 7 (*H-Frame* micro-frame 7 or 0). When an *H-Frame* boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list. (e.g. it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer, the use of which is described below.

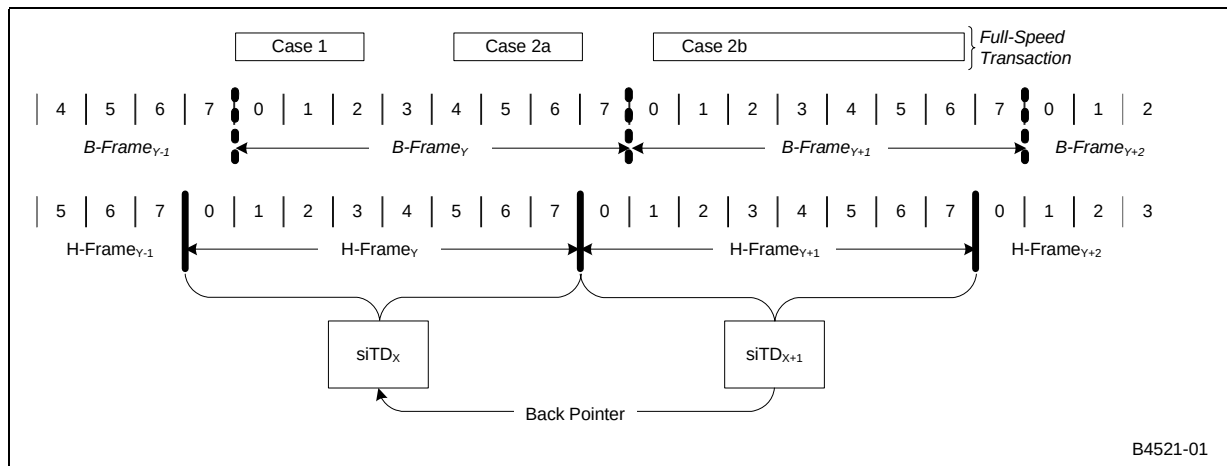
Software must never schedule full-speed isochronous OUTs across an *H-Frame* boundary.



- **Case 2b:** This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579-byte maximum packet size. A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol.
- **SplitXState.** This is a single bit residing in the *Status* field of an siTD ([Table 153, “siTD Transfer Status and Control” on page 397](#)). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in [“Split Transaction Execution State Machine for Isochronous” on page 473](#).
- **Frame S-mask.** This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an *H-Frame*) that the host controller should execute a start-split transaction. This is always qualified by the value of the *SplitXState* bit. For example, referring to the IN example in [Figure 72, “Split Transaction, Isochronous Scheduling Boundary Conditions” on page 469](#), case one, the *S-mask* would have a value of 00000001b indicating that if the siTD is traversed by the host controller, and the *SplitXState* indicates **Do Start Split**, and the current micro-frame as indicated by *FRINDEX*[2:0] is 0, then execute a start-split transaction.
- **Frame C-mask.** This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an *H-Frame*) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the *SplitXState* bit. For example, referring to the IN example in [Figure 72, “Split Transaction, Isochronous Scheduling Boundary Conditions” on page 469](#), case one, the *C-mask* would have a value of 00111100b indicating that if the siTD is traversed by the host controller, and the *SplitXState* indicates **Do Complete Split**, and the current micro-frame as indicated by *FRINDEX*[2:0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- **Back Pointer.** This field in a siTD is used to complete an IN split-transaction using the previous *H-Frame*'s siTD. This is only used when the scheduling of the complete-splits span an *H-Frame* boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN and OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the *H-Frame* boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. [Figure 73](#) illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the *B-Frames* (HS/FS/LS Bus) and the *H-Frames*. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each *H-Frame* corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

**Figure 73. siTD Scheduling Boundary Examples**

Each case is described below:

- **Case 1:** One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single *H-Frame*.
- **Case 2a, 2b:** Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction: siTD_X is used to always issue the start-split and the first *N* complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of *H-Frame*_{Y+1}, or micro-frame 0 of *H-Frame*_{Y+2}. The complete splits are scheduled using siTD_{X+2} (not shown). The complete-splits to extract this data must use the buffer pointer from siTD_{X+1}. The only way for the host controller to reach siTD_{X+1} from *H-Frame*_{Y+2} is to use siTD_{X+2}'s back pointer. The host controller rules for when to use the back pointer are described in [“Periodic Isochronous - Do Complete Split”](#) on page 476.

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so that it will complete before the end of the *B-Frame*.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in *H-Frame*, *micro-frame* 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of *H-Frame* *N* and the last complete-split would need to occur in micro-frame 1 of *H-Frame* *N*+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

9.14.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where device to host data is lost. Isochronous endpoints do not employ the concept of a halt on error, however the host is required to identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.



In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs, for their transfers, and the data structures are only reachable via the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD reinitialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD via the fields *Transaction Position (TP)* and *Transaction Count (T-count)*. If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See [“Periodic Isochronous - Do Start Split” on page 474](#) for a description on how these fields are used during a sequence of start-split transactions.

The fields *siTD.T-Count* and *siTD.TP* are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, *S-mask*, *T-Count*, and *TP* initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- *C-prog-mask*. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. *C-prog-mask* is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2:0]) number in which the complete-split was executed. The host controller always checks *C-prog-mask* before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's *Active* bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. Refer to [“Periodic Isochronous - Do Complete Split” on page 476](#) for a description on how the state of the transfer is advanced. It is important to note that an IN siTD is retired based solely on the responses from the Transaction Translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD field *Total Bytes to Transfer* to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (e.g. high-speed data streams through queue heads), the transition of *Total Bytes to*



Transfer to zero signals the end of the transfer and results in setting of the *Active* bit to zero. However, in this case, the result has not been delivered by the Transaction Translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a Transaction Translator (see Chapter 11 of the Universal Serial Bus Revision 2.0). In summary the periodic pipeline rules require that on a micro-frame boundary, the Transaction Translator will hold the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and give the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the Transaction Translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the Transaction Translator will respond with an MDATA and send all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement its *Total Bytes to Transfer* field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the Transaction Translator (for example, the Transaction Translator may have detected a CRC failure, and this result must be forwarded to the host).

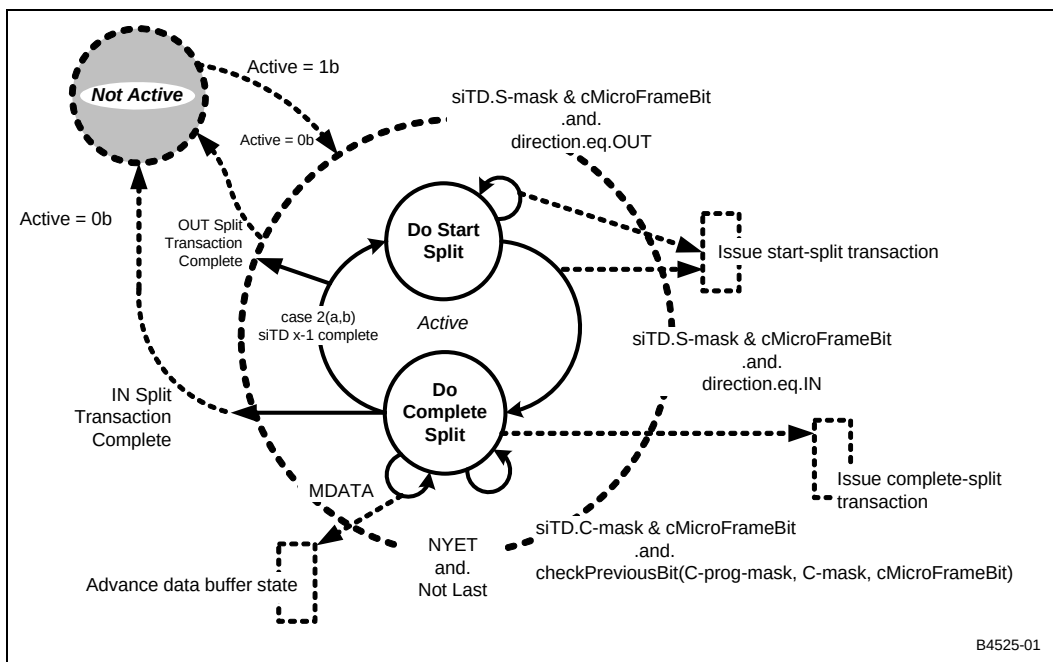
If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator will not be consistent and the transaction translator will detect and react to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the *C-prog-mask* is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (e.g. a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (e.g. state not advanced) and report the appropriate error to the client driver.

9.14.12.3.3 Split Transaction Execution State Machine for Isochronous

In the following presentation, all references to micro-frame are in the context of a micro-frame within an *H-Frame*.

If the *Active* bit in the Status byte is a zero, the host controller will ignore the siTD and continue traversing the periodic schedule. Otherwise the host controller will process the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in [“Tracking Split Transaction Progress for Isochronous Transfers” on page 471](#), plus the variable *cMicroFrameBit* defined in [“Split Transaction Execution State Machine for Interrupt” on page 461](#) to track the progress of an isochronous split transaction. [Figure 74, “Split Transaction State Machine for Isochronous” on page 474](#) illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the *Active* bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

Figure 74. Split Transaction State Machine for Isochronous



Periodic Isochronous - Do Start Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is either initialized to this state, or the siTD transitions to this state from **Do Complete Split** when a case 2a (IN) or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the *siTD.S-mask* against *cMicroFrameBit*. If there is a one in the appropriate position, the siTD will execute a start-split transaction. By definition, the host controller cannot reach an siTD at the wrong time. If the *I/O* field indicates an IN, then the start-split transaction includes only the extended token plus the full-speed token. Software must initialize the *siTD.Total Bytes To Transfer* field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (i.e. the *I/O* field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating *siTD.Current Offset* with the page pointer indicated by the page selector field (*siTD.P*). A zero in this field selects Page 0 and a 1 selects Page 1. During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the *siTD.P*-bit from a zero to a one, and begin using the *siTD.Page 1* with *siTD.Current Offset* as the memory address pointer. The field *siTD.TP* is used to annotate each start-split transaction with the indication of which part of the split-transaction data the current payload represents (ALL, BEGIN, MID, END). In all cases the host controller simply uses the value in *siTD.TP* to mark the start-split with the correct transaction position code.



T-Count is always initialized to the number of start-splits for the current frame. *TP* is always initialized to the first required transaction position identifier. The scheduling boundary case (see Figure 73, “siTD Scheduling Boundary Examples” on page 471) is used to determine the initial value of *TP*. The initial cases are summarized in Table 181.

Table 181. Initial Conditions for OUT siTD's TP and T-count Fields

Case	T-count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates *T-Count* and *TP* appropriately so that the next start-split is correctly annotated. Table 182 illustrates all of the *TP* and *T-count* transitions, which must be accomplished by the host controller.

Table 182. Transaction Position (TP) /Transaction Count (T-Count) Transition Table

TP	T-count Next	TP Next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when <i>T-count</i> starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when <i>T-count</i> starts at greater than 2.
MID	!=1	MID	<i>TP</i> stays at MID while <i>T-count</i> is not equal to 1 (e.g. greater than 1). This case can occur for any of the scheduling boundary cases where the <i>T-count</i> starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the <i>T-count</i> starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The *siTD.Total Bytes To Transfer* and the *siTD.Current Offset* fields are adjusted to reflect the number of bytes transferred.
- The *siTD.P* (page selector) bit is updated appropriately.
- The *siTD.TP* and *siTD.T-count* fields are updated appropriately as defined in Table 182.

These fields are then written back to the memory based siTD. The *S-mask* is fixed for the life of the current budget. As mentioned above, *TP* and *T-count* are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of *S-mask*, the actual number of start-split transactions depends on *T-count* (or equivalently, *Total Bytes to Transfer*). The host controller must set the *Active* bit to a zero when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when *T-Count* decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when *Total Bytes to Transfer* decrements to zero. Either implementation must ensure that if the initial condition is *Total Bytes to Transfer* equal to zero and *T-count* is equal to a one, then the host controller will issue a single start-split, with a zero-length data payload. Software must ensure that *TP*, *T-count* and *Total Bytes to Transfer* are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination will yield undefined behavior.



If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer will not progress appropriately. The transaction translator will observe protocol violations in the arrival of the start-splits for the OUT endpoint (i.e. the transaction position annotation will be incorrect as received by the transaction translator).

Example scenarios are described in [“Split Transaction for Isochronous — Processing Examples” on page 479](#).

A host controller implementation can optionally track the progress of an OUT split transaction by setting appropriate bits in the *siTD.C-prog-mask* as it executes each scheduled start-split. The *checkPreviousBit()* algorithm defined in [“Periodic Isochronous - Do Complete Split” on page 476](#) can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then set the siTD's *Active* bit to zero and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

Periodic Isochronous - Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the **Do Start State** after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on which micro-frame the host controller is currently executing which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- **Test A.** *cMicroFrameBit* is bit-wise anded with *siTD.C-mask* field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.
- **Test B.** The *siTD.C-prog-mask* bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is below (this is slightly different than the algorithm used in [“Periodic Interrupt - Do Complete Split” on page 464](#)). The sequence in which this test is applied depends on the current value of *FRINDEX[2:0]*. If *FRINDEX[2:0]* is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

Algorithm Boolean CheckPreviousBit (*siTD.C-prog-mask*, *siTD.C-mask*, *cMicroFrameBit*)

Begin

```
Boolean rvalue = TRUE;
```

```
previousBit = cMicroFrameBit rotate-right(1)
```

```
-- Bit-wise anding previousBit with C-mask indicates whether  
there was an intent
```

```
-- to send a complete split in the previous micro-frame. So, if the  
'previous bit' is set in C-mask, check C-prog-mask to make sure it happened.
```

```
if previousBit bitAND siTD.C-mask then
```

```
    if not (previousBit bitAND siTD.C-prog-mask) then
```

```
        rvalue = FALSE
```

```
    End if
```

```
End if
```



Return rvalue

End Algorithm

If Test A is true and FRINDEX[2:0] is zero or one, then this is a case 2a or 2b scheduling boundary (see [Figure 72, “Split Transaction, Isochronous Scheduling Boundary Conditions” on page 469](#)). See [“Periodic Isochronous - Do Complete Split” on page 476](#) for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller will execute a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates *QH.C-prog-mask* by bit-ORing with *cMicroFrameBit*. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from *siTD.Total Bytes To Transfer*,
- Adjust *siTD.Current Offset* by the number of bytes received,
- Adjust *siTD.P* (page selector) field if the transfer caused the host controller to use the next page pointer, and
- Set any appropriate bits in the *siTD.Status* field, depending on the results of the transaction.

Note that if the host controller encounters a condition where *siTD.Total Bytes To Transfer* is zero, and it receives more data, the host controller must not write the additional data to memory. The *siTD.Status.Active* bit must be set to zero and the *siTD.Status.Babble Detected* bit must be set to a one. The fields *siTD.Total Bytes To Transfer*, *siTD.Current Offset*, and *siTD.P* (page selector) are not required to be updated as a result of this transaction attempt.

The host controller must accept (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of *siTD.Total Bytes To Transfer*) MDATA and DATA0/1 data payloads up to and including 192 bytes. A host controller implementation may optionally set *siTD.Status.Active* to a zero and *siTD.Status.Babble Detected* to a one when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the *ERR* bit in the *siTD.Status* field and sets the *Active* bit to a zero.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, etc. The *siTD.Status.XactErr* field is set to a one and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the *Active* bit is set to zero.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the *Active* bit is set to a zero. If the *Bytes To Transfer* field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This *short packet* event does not set the USBINT status bit in the USBSTS register to a one. The host controller will not detect this condition.
- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in [“Periodic Interrupt - Do Complete Split” on page 464](#). If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the *Active* bit is set to a zero. No bits are



set in the *Status* field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.

- MDATA (and Last). See above description for testing for **Last**. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or software set up the *S-mask* and/or *C-masks* incorrectly. The host controller must set *XactErr* bit to a one and the *Active* bit is set to a zero.
- NYET (and not Last). See above description for testing for **Last**. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for *C-prog-mask*) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator will respond with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the *Missed Micro-Frame* status bit and sets the *Active* bit to a zero.

Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see [Figure 72, “Split Transaction, Isochronous Scheduling Boundary Conditions” on page 469](#)) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. [Table 183](#) enumerates the transaction state fields.

Table 183. Summary siTD Split Transaction State

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) T-count (transaction count)	All bits in the status field	C-prog-mask

Note: *TP* and *T-count* are used only for Host to Device (OUT) endpoints.

If software has budgeted the schedule of this data stream with a frame wrap case, then it must initialize the *siTD.Back Pointer* field to reference a valid siTD and will have the *siTD.Back Pointer.T-bit* in the *siTD.Back Pointer*

field set to a zero. Otherwise, software must set the *siTD.Back Pointer.T-bit* in the *siTD.Back Pointer* field to a one. The host controller's rules for interpreting when to use the *siTD.Back Pointer* field are listed below. These rules apply only when the siTD's *Active* bit is a one and the *SplitXState* is **Do Complete Split**.

- When *cMicroFrameBit* is a 1h and the *siTDX.Back Pointer.T-bit* is a zero, or
- If *cMicroFrameBit* is a 2h and *siTDX.S-mask[0]* is a zero

When either of these conditions apply, then the host controller must use the transaction state from *siTD_{X-1}*.



In order to access $siTD_{X-1}$, the host controller reads on-chip the siTD referenced from $siTD_X$.Back Pointer.

The host controller must save the entire state from $siTD_X$ while processing $siTD_{X-1}$. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of $siTD$.Back Pointers.

If $siTD_{X-1}$ is active (*Active* bit is a one and *SplitXStat* is **Do Complete Split**), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see [Table 183, “Summary siTD Split Transaction State” on page 478](#)) of $siTD_{X-1}$ is appropriately advanced based on the results and written back to memory. If the resultant state of $siTD_{X-1}$'s *Active* bit is a one, then the host controller returns to the context of $siTD_X$, and follows its next pointer to the next schedule item. No updates to $siTD_X$ are necessary.

If $siTD_{X-1}$ is active (*Active* bit is a one and *SplitXStat* is **Do Start Split**), then the host controller must set *Active* bit to a zero and *Missed Micro-Frame* status bit to a one and the resultant status written back to memory.

If $siTD_{X-1}$'s *Active* bit is a zero, (because it was zero when the host controller first visited $siTD_{X-1}$ via $siTD_X$'s back pointer, it transitioned to zero as a result of a detected error, or the results of $siTD_{X-1}$'s complete-split transaction transitioned it to zero), then the host controller returns to the context of $siTD_X$ and transitions its *SplitXState* to **Do Start Split**. The host controller then determines whether the case 2b start split boundary condition exists (i.e. if *cMicroframeBit* is a 1b and $siTD_X$.S-mask[0] is a 1b). If this criterion is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of $siTD_X$, then follows $siTD_X$.Next Pointer to the next schedule item. If the criterion is not met, the host controller simply follows $siTD_X$.Next Pointer to the next schedule item. Note that in the case of a 2b boundary case, the split-transaction of $siTD_{X-1}$ will have its *Active* bit set to zero when the host controller returns to the context of $siTD_X$. Also, note that software should not initialize an siTD with *C-mask* bits 0 and 1 set to a one and an *S-mask* with bit zero set to a one. This scheduling combination is not supported and the behavior of the host controller is undefined.

9.14.12.3.4 Split Transaction for Isochronous — Processing Examples

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced via the **Execute Transaction** queue head traversal state machine (see [Figure 65, “Host Controller Queue Head Traversal State Machine” on page 442](#)).

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, [Table 184](#) illustrates a couple of frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.



Table 184. Example Case 2a - Software Scheduling siTDs for an IN Endpoint

siTDx		Micro-Frames								Initial SplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, *S-masks* of all siTDs for this endpoint have a value of 10h (a one bit in micro-frame 4) and *C-mask* value of C3h (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the *Back Pointer* field of each siTD references the appropriate siTD data structure (and the *Back Pointer T-bits* are set to zero).

The initial *SplitXState* of the first siTD is **Do Start Split**. The host controller will visit the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is **Do Start Split**. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes *SplitXState* to **Do Complete Split**. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has its *SplitXState* initialized to **Do Complete Split**. As the host controller continues to traverse the schedule during *H-Frame* X+1, it will visit the second siTD eight times. During micro-frames 0 and 1 it will detect that it must execute complete-splits.

During *H-Frame* X+1, micro-frame 0, the host controller detects that siTD_{X+1}'s *Back Pointer.T-bit* is a zero, saves the state of siTD_{X+1} and fetches siTD_X. It executes the complete split transaction using the transaction state of siTD_X. If the siTD_X split transaction is complete, siTD's *Active* bit is set to zero and results written back to siTD_X. The host controller retains the fact that siTD_X is retired and transitions the *SplitXState* in the siTD_{X+1} to **Do Start Split**. At this point, the host controller is prepared to execute the start-split for siTD_{X+1} when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in "[Periodic Isochronous - Do Complete Split](#)" on page 476), i.e. before all the scheduled complete-splits have been executed, the host controller will transition siTD_X.*SplitXState* to **Do Start Split** early and naturally skip the remaining scheduled complete-split transactions. For this example, siTD_{X+1} does not receive a DATA0 response until *H-Frame* X+2, micro-frame 1.

During *H-Frame* X+2, micro-frame 0, the host controller detects that siTD_{X+2}'s *Back Pointer.T-bit* is a zero, saves the state of siTD_{X+2} and fetches siTD_{X+1}. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the *Active* bit. The host controller returns to the context of siTD_{X+2}, and traverses its next pointer without any state change updates to siTD_{X+2}. S

During *H-Frame* X+2, micro-frame 1, the host controller detects siTD_{X+2}'s *S-mask[0]* is a zero, saves the state of siTD_{X+2} and fetches siTD_{X+1}. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and sets the



Active bit to a zero. It returns to the state of $siTD_{x+2}$ and changes its *SplitXState* to **Do Start Split**. At this point, the host controller is prepared to execute start-splits for $siTD_{x+2}$ when it reaches micro-frame 4.

9.14.13 Host Controller Pause

When the host controller's *HCHalted* bit in the USBSTS register is a zero, the host controller is sending SOF (Start OF Frame) packets down all enabled ports. When the schedules are enabled, the EHCI host controller will access the schedules in main memory each micro-frame. This constant ping-pong of main memory is known to create CPU power management problems for mobile systems. Specifically, mobile systems aggressively manage the state of the CPU, based on recent history usage. In the more aggressive power saving modes, the CPU can disable its caches. Current PC architectures assume that bus-master accesses to main memory must be cache-coherent. So, when bus masters are busy touching memory, the CPU power management software can detect this activity over time and inhibit the transition of the CPU into its lowest power savings mode. USB controllers are bus-masters and the frequency at which they access their memory-based schedules keeps the CPU power management software from placing the CPU into its lowest power savings state.

USB Host controllers don't access main memory when they are suspended. However, there are a variety of reasons why placing the USB controllers into suspend won't work, but they are beyond the scope of this document. The base requirement is that the USB controller needs to be kept out of main memory, while at the same time, the USB bus is kept from going into suspend.

EHCI controllers provide a large-grained mechanism that can be manipulated by system software to change the memory access pattern of the host controller. System software can manipulate the schedule enable bits in the USBCMD register to turn on/off the scheduling traversal. A software heuristic can be applied to implement an on/off duty cycle that allows the USB to make reasonable progress and allow the CPU power management to get the CPU into its lowest power state. This method is not intended to be applied at all times to throttle USB, but should only be applied in very specific configurations and usage loads. For example, when only a keyboard or mouse is attached to the USB, the heuristic could detect times when the USB is attempting to move data only very infrequently and can adjust the duty cycle to allow the CPU to reach its low power state for longer periods of time. Similarly, it could detect increases in the USB load and adjust the duty cycle appropriately, even to the point where the schedules are never disabled. The assumption here is that the USB is moving data and the CPU will be required to process the data streams.

It is suggested that in order to provide a complete solution for the system, the companion host controllers should also provide a similar method to allow system software to inhibit the companion host controller from accessing its shared memory based data structures (schedule lists or otherwise).

9.14.14 Port Test Modes

EHCI host controllers must implement the port test modes **Test J_State**, **Test K_State**, **Test_Packet**, **Test_Force_Enable**, and **Test_SE0_NAK** as described in the USB Specification Revision 2.0. The system is only allowed to test ports that are owned by the EHCI controller (e.g. *CF-bit* is a one and *PortOwner* bit is a zero). System software is allowed to have at most one port in test mode at a time. Placing more than one port in test mode will yield undefined results. The required, per port test sequence is (assuming the *CF-bit* in the CONFIGFLAG register is a one):

- Disable the periodic and asynchronous schedules by setting the *Asynchronous Schedule Enable* and *Periodic Schedule Enable* bits in the USBCMD register to a zero.



- Place all enabled root ports into the suspended state by setting the *Suspend* bit in each appropriate PORTSC register to a one.
- Set the *Run/Stop* bit in the USBCMD register to a zero and wait for the *HCHalted* bit in the USBSTS register, to transition to a one. Note that an EHCI host controller implementation may optionally allow port testing with the *Run/Stop* bit set to a one. However, all host controllers must support port testing with *Run/Stop* set to a zero and *HCHalted* set to a one.
- Set the *Port Test Control* field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is **Test_Force_Enable**, then the *Run/Stop* bit in the USBCMD register must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (*HCHalted* bit is a one) then it terminates and exits test mode by setting *HCReset* to a one.

9.14.15 Interrupts

The EHCI Host Controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.), and
- Host Controller error events

All transaction-based sources are maskable through the Host Controller's Interrupt Enable register (USBINTR, see [Section 9.12.3, "USBINTR" on page 377](#)). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the *Interrupt Threshold Control* field in the USBCMD register. The value of this register controls when the host controller will generate an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller will not generate interrupts any more frequently than once every eight micro-frames.

["Host System Error" on page 485](#) details effects of a host system error.

If an interrupt has been scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to host memory. This may sometimes result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to read the USBSTS (USB Status Register). It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure



call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

Note: the host controller is not required to de-assert a currently active interrupt condition when software sets the interrupt enables (in the USBINR register, see [Section 9.12.3, “USBINTR” on page 377](#)) to a zero. The only reliable method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register ([Section 9.12.2, “USBSTS” on page 375](#)) from a one to a zero.

9.14.15.1 Transfer / Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

9.14.15.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. [Table 185](#) lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the *XactErr* status bit in the appropriate interface data structure.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the *XactErr* status bit in the queue head is set and the *CErr* field is decremented. When the *PIDCode* indicates a SETUP, the following responses are protocol errors and result in *XactErr* bit being set to a one and the *CErr* field being decremented.

- *EPS* field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- *EPS* field indicates a high-speed device and it returns a Nyet handshake to a SETUP.
- *EPS* field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

Table 185. Summary of Transaction Errors

Event / Result	Queue Head / qTD / iTD / siTD Side-effects		USB Status Register (USBSTS)
	CErr	Status Field	USBERRINT
CRC	-1	XactErr set to a one.	1 ¹
Timeout	-1	XactErr set to a one.	1 ¹
Bad PID ²	-1	XactErr set to a one.	1 ¹
Babble	N/A	“Serial Bus Babble” on page 483	1
Buffer Error	N/A	“Data Buffer Error” on page 484	
Notes: 1. If occurs in a queue head, then USBERRINT is asserted only when CErr counts down from a one to a zero. In addition the queue is halted, see “Halting a Queue Head” on page 447 . 2. The host controller received a response from the device, but it could not recognize the PID as a valid PID.			

Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a *Packet Babble*. When a device sends more data than the *Maximum Length* number of bytes, the host controller sets the *Babble Detected* bit to a one and halts the endpoint if it is using a



queue head (see [“Halting a Queue Head” on page 447](#)). *Maximum Length* is defined as the minimum of *Total Bytes to Transfer* and *Maximum Packet Size*. The *CErr* field is not decremented for a packet babble condition (only applies to queue heads). A babble condition also exists if IN transaction is in progress at High-speed EOF2 point. This is called a frame babble. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port to which the frame babble is detected.

The *USBERRINT* bit in the *USBSTS* register is set to a one and if the *USB Error Interrupt Enable* bit in the *USBINTR* register is a one, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that will babble across a micro-frame EOF.

Note: When a host controller detects a data PID mismatch, it must either: disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on *Maximum Packet Size*. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (e.g. expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence. The EHCI interface allows System software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device will re-send its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

Data Buffer Error

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors, and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the *Data Buffer Error* bit in the queue head, *iTD* or *siTD*.

If the data buffer error occurs on a non-isochronous IN, the host controller will not issue a handshake to the endpoint. This will force the endpoint to resend the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable. An acceptable implementation option is to 1's complement the CRC bytes and send them. There are other options suggested in the Transaction Translator section of the USB Specification Revision 2.0.

9.14.15.1.2 USB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (*iTDs*, *siTDs*, and queue heads (*qTDs*)) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes the USB Interrupt (*USBINT*) bit in the *USBSTS* register to be set to a one. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes *USBINT* to be set to a one. If the USB Interrupt Enable bit in the *USBINTR* register is set to a one, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, the *USBERRINT* bit in the *USBSTS* register is also set to a one.



9.14.15.1.3 Short Packet

Reception of a data packet that is less than the endpoint's Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer. Whenever a short packet completion occurs during a queue head execution, the *USBINT* bit in the *USBSTS* register is set to a one. If the *USB Interrupt Enable* bit is set in the *USBINTR* register, a hardware interrupt is signaled to the system at the next interrupt threshold.

9.14.15.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance, see [“Interrupt on Async Advance” on page 485](#)).

9.14.15.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set to a one, the host controller sets the *Port Change Detect* bit in the *USBSTS* register to a one. If the *Port Change Interrupt Enable* bit in the *USBINTR* register is a one, then the host controller will issue a hardware interrupt. The port status change bits include:

- Connect Status Change
- Port Enable/Disable Change
- Over-current Change
- Force Port Resume

9.14.15.2.2 Frame List Rollover

This event indicates that the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. If the frame list size is 1024, then the interrupt will occur every 1,024 ms, if it is 512, then it will occur every 512 ms, etc. When a frame list rollover is detected, the host controller sets the *Frame List Rollover* bit in the *USBSTS* register to a one. If the *Frame List Rollover Enable* bit in the *USBINTR* register is set to a one, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

9.14.15.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. Whenever the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of the *Interrupt on Async Advance Doorbell* bit in the *USBCMD* register. If it is a one, it sets the *Interrupt on Async Advance* bit in the *USBSTS* register to a one. If the *Interrupt on Async Advance Enable* bit in the *USBINTR* register is a one, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in [Section 9.14.8.2, “Removing Queue Heads from Asynchronous Schedule” on page 432](#).

9.14.15.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error may be catastrophic to the host controller (such as a Master Abort) making it impossible for the host controller to continue in a coherent fashion. In the presence of non-catastrophic host errors, such as parity errors, the host controller could potentially continue operation. The recommended behavior for these types of errors is to escalate it to a catastrophic error and halt the host controller. Host-based error must result in the following actions:

- The *Run/Stop* bit in the *USBCMD* register is set to a zero.
- The following bits in the *USBSTS* register are set:



- *Host System Error* bit is to a one.
- *HCHalted* bit is set to a one.
- If the *Host System Error Enable* bit in the USBINTR register is a one, then the host controller will issue a hardware interrupt. This interrupt is not delayed to the next interrupt threshold. Table 186 summarizes the required actions taken on the various host errors.

Table 186. Summary Behavior of EHCI Host Controller on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal †
siTD fetch (read)	Fatal	Fatal	Fatal †
siTD status write-back (write)	Fatal †	Fatal †	Fatal †
iTD fetch (read)	Fatal	Fatal	Fatal †
iTD status write-back (write)	Fatal †	Fatal †	Fatal †
qTD fetch (read)	Fatal	Fatal	Fatal †
qHD status write-back (write)	Fatal †	Fatal †	Fatal †
Data write	Fatal †	Fatal †	Fatal †
Data read	Fatal	Fatal	Fatal †
† Potentially, a host controller implementation could continue operation without a halt. However, the recommended behavior is to halt the host controller.			

Note: After a *Host System Error*, Software must reset the host controller via *HCRreset* in the USBCMD register before re-initializing and restarting the host controller.

9.15 EHCI Deviation

For the purposes a dual-role Host/Device controller with support for On-The-Go applications, it is necessary to deviate from the EHCI specification. Device operation and On-The-Go operation is not specified in the EHCI and thus the implementation supported in this core is proprietary. The host mode operation of the core is near EHCI compatible with few minor differences documented in this section.

The particulars of the deviations occur in the areas summarized here:

- Embedded Transaction Translator — Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation — In host mode the device operational registers are generally disabled and thus device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- Embedded design interface — This core does not support a PCI Interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.
- On-The-Go Operation — This design includes an On-The-Go controller for Port #1.

9.15.1 Embedded Transaction Translator Function

The OTG controller supports directly connected full and low speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high speed hub transaction translator. Although there is no separate Transaction Translator block in



the system, the transaction translator function normally associated with a high speed hub has been implemented within the DMA and Protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models that exist in the EHCI specification to support full and low speed devices.

9.15.1.1 Capability Registers

The following additions have been added to the capability registers to support the embedded Transaction Translator Function:

- N_TT added to Table 131, “HCSPARAMS – Host Control Structural Parameters” on page 371.
- N_PTT added to Table 131, “HCSPARAMS – Host Control Structural Parameters” on page 371.

See Section 9.11.3, “HCSPARAMS – EHCI Compliant with Extensions” on page 370 with extensions for usage information.

9.15.1.2 Operational Registers

The following additions have been added to the operational registers to support the embedded TT:

- A new register.
- Addition of two-bit Port Speed (PSPD) to the PORTSCx register.

9.15.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a Full speed (FS) or Low speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable will only be set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (i.e. Chirp completes successfully).

Since this controller has an embedded Transaction Translator, the port enable will always be set after the port reset operation regardless of the result of the host device chirp result and the resulting port speed will be indicated by the PSPD field in PORTSCx.

Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs.

The change is a fundamental one in that is summarized in the next table.

Table 187. Standard EHCI vs. EHCI with Embedded Transaction Translator

Standard EHCI	EHCI with Embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSCx.
FS and LS devices are assumed to be downstream from a HS hub thus, all port-level control is performed through the Hub Class to the nearest hub.	FS and LS device can be either downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done using the Hub Class through the nearest hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSCx.
FS and LS devices are assumed to be downstream from a HS hub with HubAddr=X. [where HubAddr > 0 and HubAddr is the address of the hub where the bus transitions from HS to FS/LS (that is, Split target hub)]	FS and LS device can be either downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = 0 and HubAddr is the address of the Root hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)]



9.15.1.4 Data Structures

The same data structures used for FS/LS transactions through a HS hub are also used for transactions through the Root hub with an embedded Transaction Translator. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

- QH (for direct attach FS/LS) – Async. (Bulk/Control Endpoints) Periodic (Interrupt)
 - Hub Address = 0
 - Transactions to direct attached device/hub.
- QH.EPS = Port Speed
 - Transactions to a device downstream from direct attached FS hub.
- QH.EPS = Downstream Device Speed

Note: When QH.EPS = 01 (LS) and PORTSCx.PSPD = 00 (FS), a LS-pre-pid will be sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

- siTD (for direct attach FS) – Periodic (ISO Endpoint)
 - All FS ISO transactions:
- Hub Address = 0
- siTD.EPS = 00 (full speed)

Maximum Packet Size must be less than or equal to 1,023 or undefined behavior may result.

9.15.1.5 Operational Model

The operational models are well defined for the behavior of the Transaction Translator (see USB 2.0 specification) and for the EHCI controller moving packets between system memory and a USB-HS hub. Since the embedded Transaction Translator exists within the host controller there is no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections will briefly discuss the operational model for how the EHCI and Transaction Translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 Transaction Translator operational models.

9.15.1.5.1 Micro-Frame Pipeline

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the Host (H) and the Bus (B). The embedded Transaction Translator uses the same pipeline algorithms specified in the USB 2.0 specification for a hub-based Transaction Translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the micro-frame pipeline implemented in the embedded Transaction Translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 will be ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded Transaction Translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream hub-based Transaction Translators.



Once periodic transfers are exhausted, any stored asynchronous transfer will be moved. Asynchronous transfers are opportunistic in that they execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer cannot babble through the SOF (start of B-frame 0.)

9.15.1.5.2 Split-State Machines

The start and complete split operational model differs from EHCI slightly because there is no bus medium between the EHCI controller and the embedded Transaction Translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete split operation is simple an internal operation to the embedded Transaction Translator. Table 188 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 188. Condition vs. Emulate TT Response

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full.	NAK
Start-Split: All periodic buffers full.	ERR
Start-Split: Success for start of Async. Transaction.	ACK
Start-Split: Start Periodic Transaction.	No Handshake (Ok)
Complete-Split: Failed to find transaction in queue.	Bus Time Out
Complete-Split: Transaction in Queue is Busy.	NYET
Complete-Split: Transaction in Queue is Complete.	[Actual Handshake from LS/FS device]

The unshaded cells represent Start-Splits and the shaded cells represent Complete-Splits.

9.15.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

USB 2.0 – 11.17.3

- Sequencing is provided & a packet length estimator ensures no full-speed/low-speed packet babbles into SOF time.

USB 2.0 – 11.17.4

- Transaction tracking for 2 data pipes.

USB 2.0 – 11.17.5

- Clear_TT_Buffer capability provided though the use of the register.

9.15.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

USB 2.0 – 11.18.6.[1-2]

- Abort of pending start-splits
 - EOF (and not started in micro-frames 6)
 - Idle for more than 4 micro-frames



- Abort of pending complete-splits
 - EOF
 - Idle for more than 4 micro-frames

USB 2.0 – 11.18.[7-8]

- Transaction tracking for up to 16 data pipes.
 - Some applications may not require transaction tracking up to a maximum of 16 periodic data pipes. The option to limit the tracking to only 4 periodic data pipes exists in the by changing the configuration constant VUSB_HS_TT_PERIODIC_CONTEXTS to 4. The result is a significant gate count savings to the core given the limitations implied.

Warning: Limiting the number of tracking pipes in the EMBEDDED –TT to four will impose the restriction that no more than four periodic transactions (INTERRUPT/ISOCRONOUS) can be scheduled through the embedded-tt per frame. the number 16 was chosen in the USB specification because it is sufficient to ensure that the high-speed to full-speed periodic pipeline can remain full. Keeping the pipeline full puts no constraint on the number of periodic transactions that can be scheduled in a frame and the only limit becomes the flight time of the packets on the bus.

- Complete-split transaction searching.

Note: There is no data schedule mechanism for these transactions other than the micro-frame pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 ms) or else undefined behavior may result.

9.15.1.5.5 Multiple Transaction Translators

The maximum number of embedded Transaction Translators that is currently supported is one as indicated by the N_TT field in [Table 131, “HCSPARAMS – Host Control Structural Parameters” on page 371.](#)

9.15.2 Device Operation

The co-existence of a device operational controller within the host controller has little effect on EHCI compatibility for host operation except as noted in this section.

9.15.2.1 USBMODE Register

Given that the dual-role controller is initialized in neither host nor device mode, the USBMODE register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

9.15.2.2 EHCI Reserved Fields

Some of the reserved fields and reserved addresses in the capability registers and operational register are used in device mode, therefore you must adhere to the following:

- Write operations to all EHCI reserved fields (some of which are device fields) with the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the host controller must properly mask EHCI reserved fields (some of which are device fields) because fields that are used exclusively for the device are undefined in host mode.



9.15.2.3 SOF Interrupt

This SOF Interrupt used for device mode is shared as a free-running, 125-us interrupt for host mode. EHCI does not specify this interrupt but it has been added for convenience and as a potential software time base. See [Section 9.12.2, “USBSTS” on page 375](#) and [Section 9.12.3, “USBINTR” on page 377](#).

9.15.3 Embedded Design Interface

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

9.15.3.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like is provided by the Frame Adjust register in the PCI configuration registers. Starts of micro-frames are timed precisely to 125 us using the transceiver clock as a reference clock. For example, using a 60-MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or using a 30-MHz transceiver clock for 16-bit physical interfaces.

9.15.4 Miscellaneous Variations from EHCI

9.15.4.1 Programmable Physical Interface Behavior

This design supports multiple Physical interfaces which can operate in differing modes when the core is configured with software programmable Physical Interface Modes. See Configuration Constants. Software programmability allows the selection of the Physical interface part during the board design phase instead of during the chip design phase. The control bits for selecting the Physical Interface operating mode have been added to the PORTSCx register providing a capability that is not defined by EHCI.

9.15.4.2 Discovery

9.15.4.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the PORTSCx register for a duration of 10ms. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10ms reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software will write a '1' to the reset the device.
- Software will write a '0' to the reset the device after 10 ms.
 - This step, which is necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a '0' to the reset bit while a reset is in progress the write will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device is now operational and at this point the port speed has been determined.



9.15.4.2.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation which will re-assign the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non High-Speed devices. Therefore, the following differences are important regarding port speed detection:

- Port Owner is read-only and always reads 0.
- A 2-bit Port Speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit High Speed indicator has been added to PORTSC to signify that the port is in High-Speed vs. Full/Low Speed – *This information is redundant with the 2-bit Port Speed indicator above.*

9.15.4.3 Port Test Mode

Port Test Control mode will behave(s) the same as described in EHCI except for test-packet mode. In test packet mode, the following procedure should be used for putting the host controller into test packet mode:

1. Write Port Test Control to 0x4.
2. Create a single QH in the Asynchronous list and program the ASYNCLISTADDR to this QH. The QH should be all zeros with the following exceptions:
 - QH.HorizontalLinkPointer = this (self pointer)
 - QH.Type = "01"
 - QH.MaximumPacketLength = 64
 - QH.H = '1'
 - QH.EPS = "10"
 - QH.Next.T = 1
 - QH.Alt.Next.T = 1
 - QH.Active = 1
 - QH.TotalBytes = 53
 - QH.BufPointer0 = (0xFFFFF000 & (test_packet[0])) >> 12
 - QH.Offset = (0x0FFF & (test_packet[0]))
 - QH.BufPointer1 = QH.BufPointer0 + 1
3. Where test_packet is a byte array of length 53 with the contents: (USB 2.0 – 7.2.20):
4. char test_packet[53];

```
test_packet[0]=0x00; test_packet[1]=0x00; test_packet[2]=0x00;
test_packet[3]=0x00; test_packet[4]=0x00; test_packet[5]=0x00;
test_packet[6]=0x00; test_packet[7]=0x00; test_packet[8]=0x00;
test_packet[9]=0xAA; test_packet[10]=0xAA; test_packet[11]=0xAA;
test_packet[12]=0xAA; test_packet[13]=0xAA; test_packet[14]=0xAA;
test_packet[15]=0xAA; test_packet[16]=0xAA; test_packet[17]=0xEE;
test_packet[18]=0xEE; test_packet[19]=0xEE; test_packet[20]=0xEE;
test_packet[21]=0xEE; test_packet[22]=0xEE; test_packet[23]=0xEE;
test_packet[24]=0xEE; test_packet[25]=0xFE; test_packet[26]=0xFF;
test_packet[27]=0xFF; test_packet[28]=0xFF; test_packet[29]=0xFF;
test_packet[30]=0xFF; test_packet[31]=0xFF; test_packet[32]=0xFF;
test_packet[33]=0xFF; test_packet[34]=0xFF; test_packet[35]=0xFF;
test_packet[36]=0xFF; test_packet[37]=0x7F; test_packet[38]=0xBF;
test_packet[39]=0xDF; test_packet[40]=0xEF; test_packet[41]=0xF7;
```




```
test_packet[42]=0xFB; test_packet[43]=0xFD; test_packet[44]=0xFC;
test_packet[45]=0x7E; test_packet[46]=0xBF; test_packet[47]=0xDF;
test_packet[48]=0xEF; test_packet[49]=0xF7; test_packet[50]=0xFB;
test_packet[51]=0xFD; test_packet[52]=0x7E;
```

5. Write Run/Stop and Asynchronous Schedule Enable (USBCMD Register) to their active states. Test packets will begin being sent continuously and SOFs will be suppressed.

To stop Test packet generation, first write **Run/Stop** and **Asynchronous Schedule Enable** to their inactive states. After **HCHALTED** (USBSTS Register) is active, then the QH can be removed and **Port Test Control** can be reprogrammed because Test Packets will have ceased.

9.16 Compatibility

This USB host is compatible with the USB 2.0 standard with the exception that it does not support a High Speed electrical interface.

9.17 Power-Management Requirements

The core is a fully synchronous static design. The power used by the design is dependent on the application usage of the core. Applications that transfer more data or use a greater number of packets to be sent will consume a greater amount of power.

9.17.1 USB Power States

The USB provides a mechanism to place segments of the USB or the entire USB into a low power suspend state. USB bus powered devices are required to respond to a 3ms lack of activity on the USB bus by going into a suspend state. In the USB core, software is notified of the suspend condition via the transition in the PORTSC register, optionally an interrupt can be generated which is controlled by the port change Detect Enable bit in the VSBINTR control register. Software then has 7 ms to transition a bus powered device into the suspend state. In the suspend state, a USB device has a maximum USB bus power budget of 500 μ A.

9.17.2 Host Power States

The USB Host core does not support power-down states.

9.18 Error / Abnormal Conditions

There are two error conditions of interest, ABH errors and overrun/underflow.

The second condition can occur when the host controller does not have sufficient bandwidth or latency of access to memory and the real-time processing of a packet. The USB specification comprehends this case and in the extremely unlikely event it should occur, an outbound packet is corrupted to cause a Device NAK and an inbound packet will be NAKed. If the packet is a bulk transfer, it will be retried, and if the packet is an ISO transfer, it will be permanently lost. Both conditions are consistent with the USB 2.0 specification.

The first condition is not readily comprehended by the USB specification per se as it is an implementation specific case. There are two cases where the AHB will return an error response, an out-of-range address and a parity or double bit DRAM error. In the case of an out-of-range address, the cause is a software fault and is fatal. The software should not chain control blocks to non-existent memory so an out-of-range address is a clear indication of a software bug. In the case of a parity or double bit SRAM error, the



packet processing and/or control block processing will cease, the error bit will be set, and an interrupt generated. The only recourse for an AHB error is a software commanded USB bus reset.



10.0 PCI Controller

10.1 Introduction

The Intel® IXP45X and Intel® IXP46X Product Line of Network Processors contains a 32-bit, 66-MHz PCI interface compatible with PCI Version 2.2.

10.2 Overview

The PCI interface is capable of operating as either a host or as an option. The PCI Controller supports these modes of operation by enabling access to the configuration register space of the IXP45X/IXP46X network processors, from either the Intel XScale® Processor when operating as a host or from an external PCI device using the PCI bus when configured as an option. Whether configured as a PCI host or PCI option, initiator or target operations are fully supported by the PCI interface of the IXP45X/IXP46X network processors.

When the PCI Controller is configured as a host, an internal PCI arbiter may be utilized to allow up to four devices to be connected to the IXP45X/IXP46X network processors without the need for an external arbiter. However, even though the internal PCI arbiter exists, the internal PCI arbiter is not required to be used when the PCI controller is configured in host or for that matter option mode of operation. The arbiter functionality is completely independent from the PCI mode of operation. An example connection of this configuration is contained in [Figure 75](#). The PCI arbiter function will allow access to the PCI bus in a round-robin fashion.

The PCI controller operating as an initiator can generate Memory, I/O, or Configuration PCI bus cycles. Operating as a target, the PCI Controller can accept Memory, I/O, or Configuration PCI bus cycles. The PCI Controller also contains two DMA engines to allow data movement between the PCI bus and the DDR1 SDRAM without the aid of the Intel XScale processor.

When the PCI Controller is configured as an option, the internal PCI arbiter can be disabled and REQ0/GNT0 are used to connect to an external arbiter on the PCI bus. An example of the IXP45X/IXP46X network processors connected in this configuration is shown in [Figure 76](#). The PCI arbiter can be enabled/disabled independently from the PCI host/option configuration.

Figure 75. PCI Bus Configured as a Host

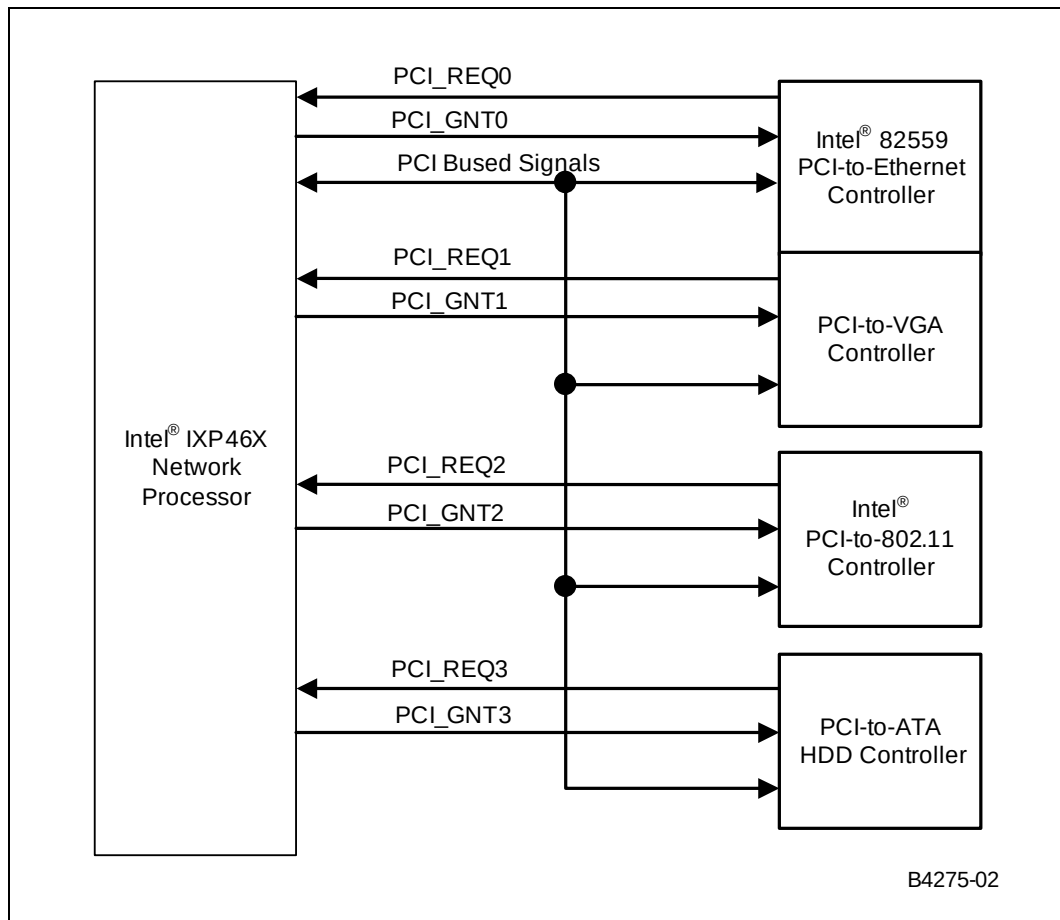
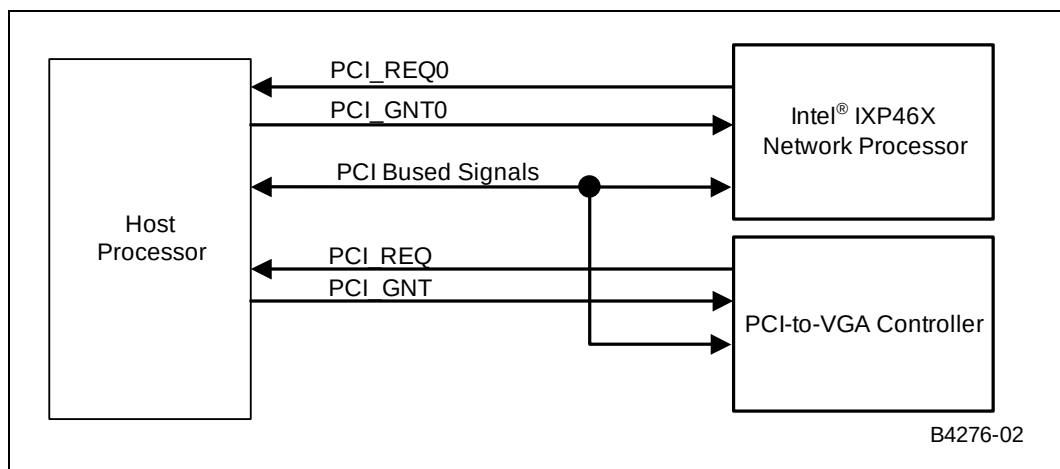


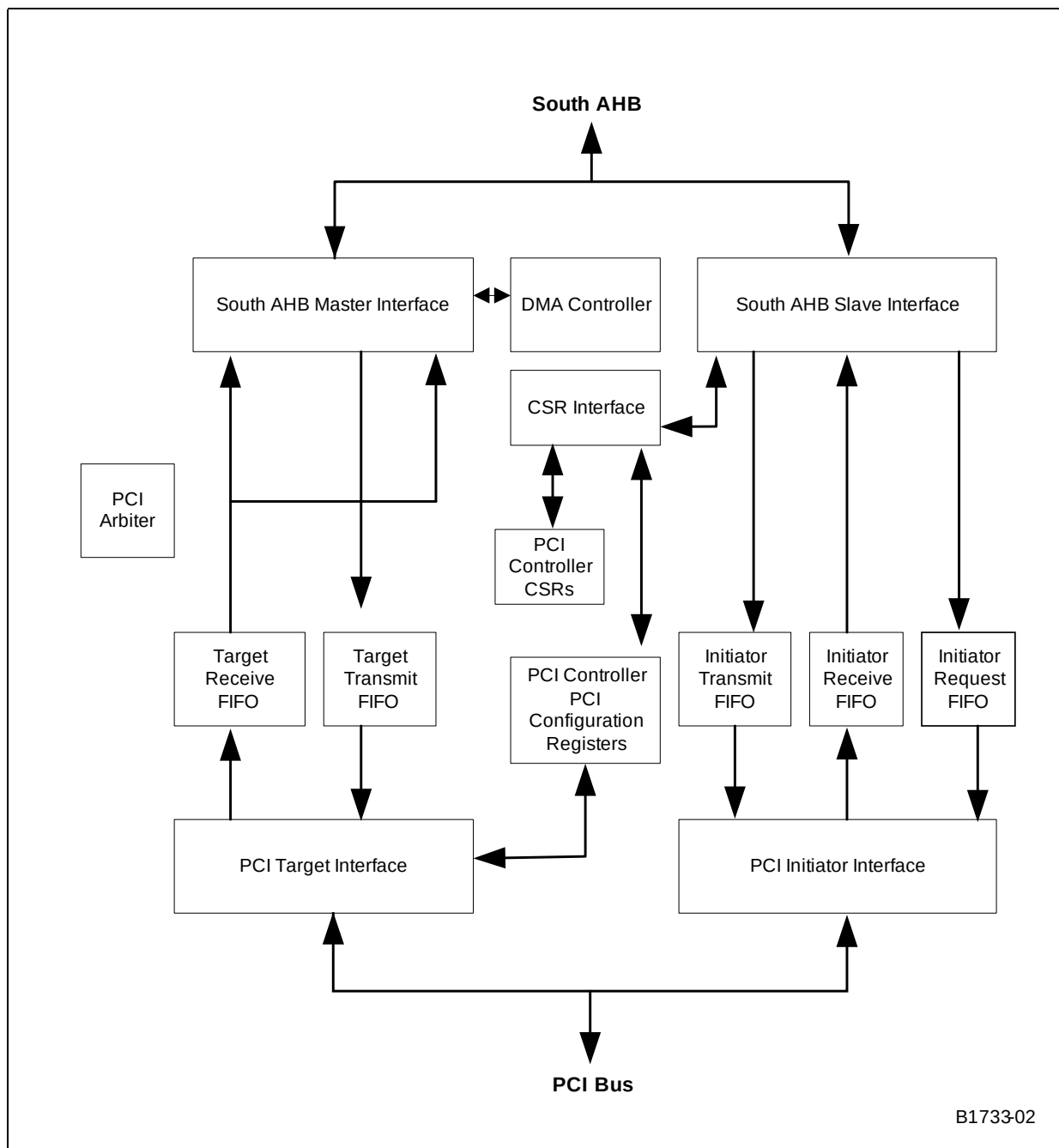
Figure 76. PCI Bus Configured as an Option



The IXP45X/IXP46X network processors PCI Controller block diagram is shown in Figure 77.



Figure 77. PCI Controller Block Diagram



The PCI Controller has two main interfaces that allow interconnection between the external PCI bus and the South AHB. The external PCI bus has both a target and an initiator controller that interfaces to the PCI bus.

When an external PCI device wants to use the IXP45X/IXP46X network processors as the target of a PCI transfer, the PCI Controller Target interface will interpret the data and forward the appropriate information (data/address/control) to the Target interface



FIFOs. The Target Interface FIFOs are 8 words deep. The PCI-target interface — in conjunction with the target interface FIFOs — will use the South AHB Master interface of the PCI Controller to provide read and write access to AHB agents, PCI Controller Configuration and status registers. The PCI Controller PCI Configuration Registers are accessed through configuration cycles. Target transactions will be accessed directly from the PCI-Target Interface.

Table 189 lists the supported command types, when the IXP45X/IXP46X network processors are used as a target of a PCI transaction. The PCI Target interface does not support the following features:

- Lock cycles
- VGA palette snoop
- Dual address cycles
- Cache-line wrap mode addressing (Disconnected after first data phase of transaction)
- Type 1 Configuration space

Table 189. PCI Target Interface Supported Commands

PCI Command/Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Not Supported
0x1	Special Cycle	Not Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Converted To Memory Read
0xD	Dual Address Cycle	Not Supported
0xE	Memory Read Line	Converted To Memory Read
0xF	Memory Write and Invalidate	Converted To Memory Write

When the IXP45X/IXP46X network processors want to use an external PCI device as the target of a PCI transfer, the PCI Controller Initiator interface will be used to generate the appropriate PCI bus cycles and forward the information to the PCI bus. There are three ways in which PCI bus cycles may be initiated:

- The DMA channels generate PCI Memory cycles. Refer to [“PCI Controller DMA” on page 536](#) for additional details.
- AHB masters generate PCI Memory cycles using memory-mapped direct access on the AHB bus. Refer to [“Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register” on page 507](#) for additional details.



- Interrupt Acknowledge, Special Cycle, I/O, Configuration, and single-data-phase Memory cycles are generated indirectly by AHB masters using a Non-pre-fetch CSR mechanism. Refer to [“PCI Controller Configured as Host” on page 501](#) for additional details.

For PCI bus memory cycles, the PCI Initiator interface will receive requests for PCI transfer from the PCI Controller DMA channels. For PCI bus I/O cycles and configuration cycles, the PCI Initiator interface will receive requests for PCI transfer from an AHB master (a single word PCI Bus Memory Cycle can be produced using this method).

Requests for PCI transfers using the PCI Controller Initiator interface are buffered in the Initiator Request FIFO and handled by the PCI Controller Initiator interface when appropriate. The Initiator interface will receive the appropriate transfer information from the Initiator Request FIFO; which is the address, word count, byte enables, and PCI command type.

The Initiator Request FIFO is a four-entry FIFO allowing up to four requests to be buffered. If a request is issued that generates an initiator transaction and the Initiator Request FIFO is already full, a retry will be issued to the AHB master that initiated the request.

After gathering the appropriate information, the PCI Initiator interface performs the specified transaction on the PCI bus, handling all bus protocol and any retry/disconnect situations. The data will be moved from the South AHB to the PCI bus using the Initiator Data FIFOs. The Initiator Data FIFOs are eight words deep. [Table 190](#) lists the supported PCI transaction types produced by the PCI Controller Initiator Interface of the IXP45X/IXP46X network processors.

Table 190. PCI Initiator Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Supported
0x1	Special Cycle	Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Supported
0xD	Dual Address Cycle	Not supported
0xE	Memory Read Line	Supported
0xF	Memory Write and Invalidate	Not supported

It is important to note that the target interface — and the DMA channels used for supporting the initiator interface — can contend for the use of the South AHB Master Controller. When this contention occurs, arbitration for control of South AHB Master Controller is carried out on two levels.



On the first level, the PCI Target Interface requests and the DMA requests alternate for priority access. On the first transaction the PCI Target interface would gain access to the South AHB Master Controller's services, followed by one of the DMA channels gaining access to the South AHB Master Controller's services, then the PCI Target interface would gain access to the South AHB Master Controller's services again, followed by the second DMA channel gaining access to the South AHB Master Controller's services, etc.

On the second level of arbitration for the South AHB Master Controller's services, the two DMA channels will alternate for priority access. DMA channel 0 would gain access to the South AHB Master Controller's services first, followed by DMA channel 1, and then DMA channel 0, etc. This arbitration scheme balances the high-bandwidth DMA traffic with what should be lower bandwidth PCI Target Interface traffic and is only used in cases where contention exists. For instance, if there are only PCI Target Interface requests being received by the South AHB Master Controller. The PCI Target would continually get access to the South AHB Master Controller until a DMA request is detected.

The PCI Controller also contains two configuration spaces. The PCI Controller Control and Status Register (CSR) configuration space is used to configure the PCI Controller, initiate single cycle PCI transactions using the non-pre-fetch registers, operate the DMA channels, report PCI Controller status, and allow access to the PCI Controller PCI Configuration Registers. The PCI Configuration Space is a 64-byte, PCI type-0 configuration space that supports a single function.

The PCI Configuration Space can be written or read using registers defined in the Control and Status Registers when the IXP45X/IXP46X network processors are configured as a PCI Host. An external PCI Master using PCI Configuration Cycles can write or read the PCI Configuration Space when the IXP45X/IXP46X network processors are configured as an Option. The PCI Configuration Space may be accessed by the Intel XScale processor or the PCI bus but never by both at the same time.

10.2.1 List of Features

- Conforms to PCI Local Bus Specification Revision 2.2
- 32-bit, 0-66MHz PCI bus operation
- Provides initiator (master) and target (slave) PCI interfaces
- Provides AHB-to-PCI and PCI-to-AHB DMA channels
- Includes PCI bus arbiter supporting up to 4 external PCI masters using round-robin arbitration.
- Access to PCI Configuration registers from PCI or AHB busses
- Provides interrupt to processor to indicate transaction errors on the PCI or AHB bus
- Provides interrupt to processor for DMA complete and DMA error
- Provides doorbell interrupt generation capability to PCI and AHB agents
- Byte, half word, word single reads/writes, burst word reads/writes supported on AHB and PCI busses
- Generates memory, I/O, and configuration cycles as PCI master
- Provides AHB masters with full access to the 4Gbyte PCI address space, via DMA channels and/or non-prefetch transactions.
- Provides PCI-to-AHB address translation to map PCI accesses to AHB address space



10.2.2 PCI Controller Configured as Host

The IXP45X/IXP46X network processors can be configured as a host function on the PCI bus. Configuring the IXP45X/IXP46X network processors as a host does not require the internal PCI arbiter function in the PCI Controller to be enabled.

The first step to using the PCI interface in any mode of operation is to determine the mode of operation and then configure the interface. The PCI bus mode of operation can be obtained by reading bit 0 of the PCI Controller Control and Status Register (PCI_CSR). If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the IXP45X/IXP46X network processors are required to function as an Option on the PCI bus. If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the IXP45X/IXP46X network processors are required to function as the Host on the PCI bus.

Bit 0 of the PCI Controller Control and Status Register (PCI_CSR) will be set by the logic level contained on Expansion Bus Address Bus bit 1 at the de-assertion of the reset signal supplied to the IXP45X/IXP46X network processors. The internal arbiter will be enabled/disabled based on the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of the reset signal supplied to the IXP45X/IXP46X network processors. The PCI Controller Control and Status Register (PCI_CSR) bit 1 captures the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of reset.

If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the internal arbiter is enabled on the IXP45X/IXP46X network processors. If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the internal arbiter is disabled on the IXP45X/IXP46X network processors.

Once the PCI controller has determined that the mode of operation is to be host, the IXP45X/IXP46X network processors are required to configure the rest of the PCI bus. However, before the IXP45X/IXP46X network processors can configure the rest of the PCI bus, the PCI Controller must be configured.

The Configuration and Status Registers must be initialized and the PCI Controller Configuration and Status Registers must be initialized. (For more detail on initializing the Configuration and Status Registers, see [“Initializing PCI Controller Configuration and Status Registers for Data Transactions” on page 505](#). For more detail on initializing the PCI Controller Configuration and Status Registers, see [“Initializing the PCI Controller Configuration Registers” on page 509](#).)

After the local Configuration and Status Register and PCI Controller Configuration and Status Registers have been initialized, the remainder of the PCI bus is ready to be configured by the hosting network processor. The IXP45X/IXP46X network processors will now begin to initiate configuration cycles to all of the potential devices on the PCI bus.

The order and nature in which the devices are learned is up to the individual application. However, one bit must be configured prior to initiating PCI Configuration Cycles with the IXP45X/IXP46X network processors. Bit 2 of the PCI Control Register/Status (PCI_SRCR) Register must be set to logic 1 using the methods described in [“Initializing the PCI Controller Configuration Registers” on page 509](#). The setting of bit 2 to logic 1 enables PCI bus-mastering capability.

Two types of PCI Configuration Cycles can be generated using the IXP45X/IXP46X network processors: Type 0 and Type 1 Configuration Cycles. Type 0 Configuration Cycles are used to communicate to a PCI device which is contained on the same local segment that the generator of the Configuration Cycles. Type 1 Configuration Cycles are used to communicate to a PCI device which is contained on another segment of the PCI bus other than the PCI bus segment that is generating the Configuration Cycles, i.e., a segment on the other side of a PCI bridge.



Figure 78. Type 0 Configuration Address Phase

Figure 79. Type 1 Configuration Address Phase

Configuration cycles will be produced by the IXP45X/IXP46X network processors using four 32-bit Configuration and Status Registers referred to as the Non-Pre-fetch Registers. These registers are:

- The Non-Pre-fetch Register accesses can also be used to produce memory and I/O PCI bus cycle. However, these cycles can only be single cycle accesses.

Non-Pre-fetch Read Cycles will be implemented by placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register and then placing the PCI Command Type and Byte Enables for the desired read cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register. The PCI Controller then will initiate the proper transaction on the PCI bus to read the requested data. Then the returned data is placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.

To avoid incorrect data from being read by the initiator of this transaction, retries will be issued to any AHB master that attempts to read the PCI Controller Configuration and Status Registers prior to the Non-Pre-fetch PCI read data being placed into the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register. This action assures that the next read of the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register does not contain stale data.

Non-Pre-fetch Write Cycles will be implemented by:

- Placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register



- Placing the PCI Command Type and Byte Enables for the desired write cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register
- Writing the data that is to be placed onto the PCI bus into the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register

The PCI Controller then will initiate the proper transaction on the PCI bus to place the requested write data onto the PCI bus. To avoid the write data from being corrupted by new request from an AHB master, retries will be issued to any AHB master that attempts to write the PCI Controller Configuration and Status Registers prior to the completion of the requested PCI transaction.

It is also noteworthy to mention that the PCI Controller does not interpret or manipulate the contents of the Non-Pre-fetch Registers. The address, command, byte enables, and write data are passed to the PCI bus as-is. For example, I/O read and I/O write requests must be set-up such that the byte-enables are consistent with the 2 LSBs of the address in accordance with the PCI local-bus specification.

10.2.2.1 Example: Generating a PCI Configuration Write and Read

This example examines the initializing of the Base Address Register.

1. Assume a PCI device has been located and now the Base Address Register configuration of this PCI device is going to be initialized. The first step is to write all logic 1s to the PCI Base Address Registers.
Base Address Register 0 will be located at hexadecimal offset of 0x10 when the ID_SEL of this device is active and the access is a PCI Bus Configuration Cycle. The intent of this exercise is to initialize this Base Address Register.
2. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.
This value will allow a write to a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1. This bit is set, assuming that ID_SEL for a given device on the local segment is selected using address bit 16. This value chosen for PCI_NP_AD follows the convention outlined in [Figure 78, "Type 0 Configuration Address Phase" on page 502](#).
3. Write a hexadecimal value of 0x0000000B to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.
Bits 7:4 of this register specify the byte enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be written. Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1011b signifies that a Configuration Write Cycle is being requested.
4. Write a hexadecimal value of 0xFFFFFFFF to the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register.
This write to the Configuration and Status Registers will cause a PCI Configuration Write Cycle with all byte-enables active to be initiated on the PCI bus.
5. Base Address Register 0 has been written with all logic 1s. However, only some of these bits will be set to logic 1.
Logic 1s will only be written to the bits corresponding to a given address space defined for the PCI device. For instance, assume that the PCI device being configured requires a 64-Mbyte address space for Base Address Register 0 used for memory transactions with no adverse side effects to reads. Only bits (31:26) would be written.
Now, the IXP45X/IXP46X network processors must read Base Address Register 0 to determine the Address Space, Address space type (memory or I/O), and any limitations to reading this address space.
6. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.



This value will allow a read from a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1.

This bit is set assuming that ID_SEL for a given device on the local segment is selected, using address bit 16. This device is the one that is attempting to be accessed.

7. Write a hexadecimal value of 0x0000000A to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.
Bits 7:4 of this register specify the byte-enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be read.
Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1010b signifies that a Configuration Read Cycle is being requested. This action causes the PCI Controller to initiate the read transaction.
8. The data returned will be placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.
The returned value looks like hexadecimal 0xFC000008. This value signifies that an address space of 64 Mbyte is being requested by Base Address Register 0 of the PCI device to be mapped anywhere into the PCI address map, the address space is a Memory Space, and there are no special read conditions that apply to this address space. (See the *PCI Local Bus Specification*, Rev. 2.2 for more details.)
9. Now the IXP45X/IXP46X network processors must specify the PCI address space that this Base Address Register is going to occupy. This is done by executing a Configuration Write to bits 31:26 of Base Address Register 0 — with the logical value where the address is going to reside.
Assume we want the address to reside at PCI location 0xA0000000. A Configuration Write of 0xA0000000 will be written to Base Address Register 0 of the external PCI device. No other PCI assignment can be placed between PCI addresses 0xA0000000 and 0xA3FFFFFF.

When the IXP45X/IXP46X network processors are functioning in Host mode of operation, all other PCI Configuration Registers contained on external PCI devices will be configured or used to configure the PCI Bus using PCI Configuration Read/Write Cycles produced from the IXP45X/IXP46X network processors for each device on the PCI bus. Some examples of these parameters are Base Address Register, and Grant Latencies. (For more details on exact settings/usage of these parameters for a given application, see the *PCI Local Bus Specification*, Rev. 2.2.)

The IXP45X/IXP46X network processors have now been successfully configured as a PCI host and successfully configured the PCI bus. PCI memory and PCI I/O transaction can now take place.

For more detail on generating PCI Memory and PCI I/O transactions using the IXP45X/IXP46X network processors, see [“PCI Controller Functioning as Bus Initiator” on page 512](#). For more detail on accepting PCI Memory and PCI I/O transactions using the IXP45X/IXP46X network processors, see [“PCI Controller Functioning as Bus Target” on page 521](#).

10.2.3 PCI Controller Configured as Option

The IXP45X/IXP46X network processors can be configured as an option function on the PCI bus without requiring the Internal Arbiter function in the PCI Controller to be enabled. Therefore, the Internal Arbiter can be enabled independently and the host/option configuration can be selected independently. The option function is selected similarly to the manner in which the host function is selected. (For more details, see [“PCI Controller Configured as Host” on page 501](#).)



If the IXP45X/IXP46X network processors are configured as an option, an external PCI Host will want to access the PCI Configuration Space of the IXP45X/IXP46X network processors. The PCI Host will complete these accesses using PCI Configuration Cycles. However, if the IXP45X/IXP46X network processors receive Configuration Cycles prior to being initialized, improper PCI bus configuration may occur.

To prevent this event from occurring, the IXP45X/IXP46X network processors can refuse to accept configuration cycles from an external source by programming bit 15 of the PCI Controller Control and Status (PCI_CSR) Register. Bit 15 of the PCI Controller Control and Status (PCI_CSR) Register is the Initialization Complete bit. When bit 15 is set to logic 0, the PCI Controller Target Interface will issue retries to PCI Configuration cycles. When bit 15 is set to logic 1, PCI Configuration Cycles will be accepted.

The Initialization Complete bit allows time for the IXP45X/IXP46X network processors to configure the chip prior to accepting cycles from an external PCI device. If initialization is not completed in the first 2^{25} PCI clocks after the PCI reset signal is de-asserted, the possibility exists for the external PCI Host to assume that no PCI device is resident or active at this particular IDSEL.

An access to the PCI Controller PCI Configuration Registers of the IXP45X/IXP46X network processors occurs when the PCI_IDSEL input is asserted, the PCI command field as represented by the PCI Command/Byte enable signals is a configuration read or write, $\text{PCI\_AD}[1:0] = 00$ indicating a type 0 configuration cycle, and the PCI Controller Target Interface is allowed to accept Type 0 Configuration Cycles by asserting the Initialization Complete bit. The PCI Configuration Register accessed is determined by the value contained on the $\text{PCI\_AD}[7:2]$ pins during the address phase of the PCI Configuration Transaction. Accesses to the PCI Configuration Register can be a single-word only. The PCI Controller Target Interface will disconnect any burst longer than one word.

During reads of the PCI Configuration Registers, byte-enables are ignored and the full 32-bit register value is always returned. Read accesses to unimplemented registers complete normally on the bus and return all zeroes.

During PCI Configuration Register writes, the PCI byte-enables determine the byte(s) that are written within the addressed register. Write accesses to unimplemented PCI Configuration Registers complete normally on the bus but the data is discarded. The PCI Configuration Space supported by the IXP45X/IXP46X network processors is a single-function, Type 0 configuration space. (For more information on the PCI Configuration Space and additional configuration details, see ["PCI Configuration Registers" on page 549](#) and the *PCI Local Bus Specification*, Rev. 2.2.)

10.2.4 Initializing PCI Controller Configuration and Status Registers for Data Transactions

In order to use the PCI Controller for transactions, other than single-word initiator transactions implemented by Non-Pre-fetch transactions, various registers must be set in the PCI Controller Configuration and Status Registers. The registers that must be initialized are:

- AHB Memory Base Address Register (PCI_AHBMEMBASE)
- AHB I/O Base Address Register (PCI_AHBIOBASE)
- PCI Memory Base Address Register (PCI_PCIMEMBASE).

The AHB Memory Base Address Register (PCI_AHBMEMBASE) is used to map the address of a PCI Memory Cycle Target transfers from the address of the PCI Bus to the address of the South AHB. The AHB I/O Base Address Register (PCI_AHBIOBASE) is used to map the address of a PCI I/O Cycle Target transfers from the address of the PCI



Bus to the address of the South AHB. The PCI Memory Base Address Register (PCI_PCIMEMBASE) is used to map the address of direct access PCI memory-mapped transfers from the address of the South AHB to the address of the PCI Bus.

When the IXP45X/IXP46X network processors are the target of a PCI bus transaction, the values written or read by external PCI Bus Initiators using the Base Address Registers contained within the IXP45X/IXP46X network processors must be translated to an address location within the processors. The configuration of the internal memory allocation is implemented differently for each of the Base Address Registers (BAR). The following paragraphs describe the implementation for each of the Base Address Registers.

For Base Address Registers 0 through 3 — which are used to complete PCI Bus Memory Cycles Target transactions — the AHB Memory Base Address (PCI_AHBMEMBASE) register is used to translate PCI Memory Cycle accesses to their appropriate AHB locations. The AHB Memory Base Address (PCI_AHBMEMBASE) register is used to determine the upper 8 AHB address bits when an external Initiator on the PCI bus accesses the memory spaces of the IXP45X/IXP46X network processors. The PCI Controller can be configured to support four 16-Mbyte locations for PCI Target Memory Cycle transactions using the AHB Memory Base Address (PCI_AHBMEMBASE) register and the PCI Base Address Registers.

The AHB Memory Base Address (PCI_AHBMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a PCI Base Address Register

- Bits 31:24 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 0 and the first 16-Mbyte AHB memory location (AHB base 0)
- Bits 23:16 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 1 and the second 16-Mbyte AHB memory location (AHB base 1)
- Bits 15:8 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 2 and the third 16-Mbyte AHB memory location (AHB base 2)
- Bits 7:0 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 3 and the fourth 16-Mbyte AHB memory location (AHB base 3).

Base Address Register 4 is used to complete accesses to internal PCI Controller Configuration and Status registers. (These registers are not the PCI Controller PCI Configuration Registers.) PCI Base Address Register 4 is used to decode that an access has been made to the Configuration and Status Register Space. There are no AHB cycles produced for this type of an access, as all accesses to this Base Address Register will be internal to the PCI controller. Therefore, an address translation register is not required.

For Base Address Register 5 — which is used to complete PCI bus I/O cycles — the AHB I/O Base Address (PCI_AHBIOBASE) register is used to translate I/O PCI accesses to their appropriate AHB locations. The PCI Controller can be configured to support a single 256-Byte location for PCI target I/O cycle transactions, using the AHB I/O Base Address (PCI_AHBIOBASE) register and PCI Base Address Register 5.

The AHB I/O Base Address (PCI_AHBIOBASE) register consists of a single 24-bit field. The AHB I/O Base Address (PCI_AHBIOBASE) register is used to determine the upper 24 AHB address bits, when an external initiator on the PCI bus accesses the I/O space of the IXP45X/IXP46X network processors.



10.2.4.1 Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register

The following example can be used to understand the operation of the AHB Memory Base Address Register (PCI_AHBMEMBASE), AHB I/O Base Address Register (PCI_AHBIOBASE), and PCI Memory Base Address Register (PCI_PCIMEMBASE).

1. Assume that PCI_AHBMEMBASE = 0x04010506 and PCI_AHBIOBASE = 0x000A1200.
2. Assume that the PCI Bus has gone through configuration and the Base Address Registers (BAR0 – BAR5) are set as follows:
 - BAR0 = 0xA0000000
 - BAR1 = 0xA1000000
 - BAR2 = 0xA2000000
 - BAR3 = 0xA3000000
 - BAR4 = 0xA4000000
 - BAR5 = 0xA5123400
3. An external PCI device initiates a PCI bus transfer to BAR1 of the IXP45X/IXP46X network processors. The PCI address looks like the following: PCI Address = 0xA100402C. The address placed on the South AHB is 0100402C.
 Notice that the third byte from the right, of the PCI_AHBMEMBASE = 0x04010506, is substituted for the A1 located in the fourth byte from the right of the PCI Address = 0xA100402C.
4. Next, an external PCI device initiates a PCI bus transfer to BAR3 of the IXP45X/IXP46X network processors. The PCI address looks like the following: PCI Address = 0xA3004014. The address placed on the South AHB is 06004014.
 Notice that the first byte from the right of the PCI_AHBMEMBASE = 0x04010506 is substituted for the A3 located in the fourth byte from the right of the PCI Address = 0xA3004014.
5. PCI I/O space example is an external PCI device initiates a PCI bus transfer to BAR5 of the IXP45X/IXP46X network processors. The PCI address looks like the following PCI Address = 0xA5123418. The address placed on the South AHB is 0x0A120018.
 Notice that the first three bytes from the right of the PCI_AHBIOBASE = 0x000A1200 is substituted for the A51234 located in the PCI Address = 0xA5123418.
6. The final example is an external PCI device initiates a PCI bus transfer to BAR4 of the IXP45X/IXP46X network processors. This allows access to the PCI Controller Configuration and Status Register. The PCI address looks like the following PCI Address = 0xA4000038. There is no address placed on the South AHB. This causes an access of the PCI Doorbell Register on the IXP45X/IXP46X network processors.

When the IXP45X/IXP46X network processors are the initiator of a PCI Bus transaction and desires the transaction to produce PCI Memory Transactions, the values may be written or read by providing a transfer to the PCI Memory Cycle Address Space defined for the IXP45X/IXP46X network processors. The 64-Mbyte address space defined for the PCI Memory Cycle Address Space is from AHB address location 0x48000000 to 0x4BFFFFFF.

Only four 16-Mbyte windows can be enabled. The four 16-Mbyte windows are divided among the addresses as shown in [Table 191](#).



Table 191. PCI Memory Map Allocation

Description	Starting Address	Ending Address
First 16-Mbyte window	0x48000000	0x48FFFFFF
Second 16-Mbyte window	0x49000000	0x49FFFFFF
Third 16-Mbyte window	0x4A000000	0x4AFFFFFF
Fourth 16-Mbyte window	0x4B000000	0x4BFFFFFF

The four 16-Mbyte windows translate their South AHB addresses to the PCI Bus addresses using the PCI Memory Base Address Register (PCI_PCIMEMBASE).

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a given 16-Mbyte window:

- Bits 31:24 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the first 16-Mbyte window from South AHB address 0x48000000 to 0x48FFFFFF
- Bits 23:16 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the second 16-Mbyte window from South AHB address 0x49000000 to 0x49FFFFFF
- Bits 15:8 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the third 16-Mbyte window from South AHB address 0x4A000000 to 0x4AFFFFFF
- Bits 7:0 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the fourth 16-Mbyte window from South AHB address 0x4B000000 to 0x4BFFFFFF.

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register is used to determine the upper eight PCI address bits when the IXP45X/IXP46X network processors access the memory spaces of external Targets on the PCI bus.

10.2.4.2 Example: PCI Memory Base Address Register and South-AHB Translation

The following example discusses the operation of the PCI Memory Base Address Register (PCI_PCIMEMBASE) and the South AHB translation.

1. Assume that PCI_PCIMEMBASE = 0xC3A24169.
2. The next example shows an access to the first 16-Mbyte window.
The South AHB address is for the access is 0x48123450. The address presented on the PCI bus is 0xC3123450.
3. The next example shows an access to the second 16-Mbyte window.
The South AHB address is for the access is 0x49123450. The address presented on the PCI bus is 0xA2123450.
4. The next example shows an access to the third 16-Mbyte window.
The South AHB address is for the access is 0x4A123450. The address presented on the PCI bus is 0x1123450.
5. The next example shows an access to the fourth 16-Mbyte window.
The South AHB address is for the access is 0x4B123450. The address presented on the PCI bus is 0x69123450.



10.2.5 Initializing the PCI Controller Configuration Registers

The PCI Base Address Registers along with any other pertinent PCI Configuration Registers, located in the PCI Controller PCI Configuration Register space, must be initialized by the Intel XScale processor when the IXP45X/IXP46X network processors are configured as the PCI host. The PCI Base Address Registers must be initialized by an external PCI device when the IXP45X/IXP46X network processors are configured as a PCI option.

The PCI Base Address Registers — along with any other registers in the PCI Configuration Space — will be accessed by the Intel XScale processor using three Configuration and Status Registers:

- PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register
- PCI Configuration Port Write Data (PCI_CRP_WDATA) Register
- PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.

The IXP45X/IXP46X network processors are a single-function, Type 0 Configuration space when functioning as a PCI option. For detailed information on the values to program the PCI Controller Configuration and Status Registers, see the *PCI Local Bus Specification*, Rev. 2.2.

The PCI Configuration Port Write Data (PCI_CRP_WDATA) Register is a 32-bit register that is used to place the data that is to be written into the PCI Configuration Space. The PCI Configuration Port Read Data (PCI_CRP_RDATA) Register is a 32-bit register that is used to capture the data that is returned from the PCI Configuration Space. The PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register provides the address, byte enables, and control for the read and write access to the PCI Configuration Space from the internal side of the IXP45X/IXP46X network processors.

- Bits 23:20 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the byte enables for the access to the PCI Configuration Space
These bits directly correspond to the four - byte field associated with the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register. [Table 192 on page 511](#) shows the mapping of the byte enables of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register to the byte lane fields of the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register.
 - Bits 7:2 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the address for the register access within the 64 32-bit Word PCI Configuration Space.
The 64 32-bit Word PCI Configuration Space is shown in [Table 193 on page 511](#).
 - Bits 19:16 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the command to execute on the PCI Configuration Space. The only two commands currently defined are read and write. [Table 194 on page 512](#) shows valid command codes for accessing the PCI Configuration Space. When a read command is written into the command field of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate address of the PCI Configuration register to be accessed, the data from the address requested will be returned to the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.
A master on the AHB bus can then read the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register. For example:
1. PCI_CRP_AD_CBE is written with hexadecimal 0x00300004, which causes the contents of the PCI Control Register/Status Register (PCI_SRCR) to be written into the PCI_CRP_RDATA register.



Note that bits 23:20 are set to hexadecimal 3. For read accesses, byte-enables are ignored. Bits 19:16 are set to hexadecimal 0, which denotes a read command. Bits 7:0 are set to hexadecimal 04.

2. PCI_CRP_RDATA is read by the AHB master that requested the PCI_SRCR to be returned to the PCI_CRP_RDATA register.

When a write to the PCI Configuration Space is desired, the AHB master requesting the write must update the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register with the data that is to be written to the PCI Configuration Register. Once the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register has been updated, a write command is written into the command field of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate byte enables and address of the PCI Configuration register to be accesses.

The data contained in the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register will be written to the PCI Configuration Register specified by the address and byte enables contained in the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register. For Example:

1. An AHB master that wants to write a particular PCI Configuration Register writes PCI_CRP_AD_CBE register first. Assume that the AHB master wants to write a hexadecimal value of 0x00001234 to the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register. The PCI_CRP_AD_CBE register is written with a hexadecimal 0x00010040.

Note that bits 23:20 are set to hexadecimal 0. For write accesses byte enables are active low. Bits 19:16 are set to hexadecimal 1, which denotes a write command. Bits 7:0 are set to hexadecimal 40, which addresses the PCI_RTOTTO register.

2. Next, the hexadecimal value of 0x00001234 is written to the PCI Configuration Register PCI_CRP_WDATA register, which causes the contents of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register to be written with a hexadecimal value of 0x00001234.

One more example will demonstrate the effects of the byte-enables on write accesses to the PCI Configuration Space:

1. Assume that the objective is to update the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) without updating the TRDY terminal count value of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 7:0). The Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is located at hexadecimal address 0x40. Also assume the value currently contained in the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is a hexadecimal 0x00008080.
2. The PCI_CRP_AD_CBE is written with hexadecimal 0x00D10040.
Note that bits 23:20 are set to hexadecimal D. For write accesses this allows only byte 1 to be written (bits 15:8). Bits 19:16 are set to hexadecimal 1, which denotes a write command. Bits 7:0 are set to hexadecimal 40, which addresses the PCI_RTOTTO register.
3. Assume that the AHB master wants to write a hexadecimal value of 0x0000AB00 to the second byte of the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8). The PCI_CRP_WDATA will be loaded with a value of 0x0000AB00, which causes the contents of the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) to be written with a hexadecimal value of 0x0000AB00. The value that is now contained within the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is 0x0000AB80. Notice that only one byte of data was manipulated.

Table 192 shows the PCI Byte Enables Byte Lane Mapping (accesses to the PCI Configuration Space from within the IXP45X/IXP46X network processors) when using the CRP access mechanism.

**Table 192. PCI Byte Enables Using CRP Access Method**

PCI_CRP_AD_CBE(23:20)	PCI_CRP_WDATA [31:24]	PCI_CRP_WDATA [24:16]	PCI_CRP_WDATA [15:8]	PCI_CRP_WDATA [7:0]
0000	X	X	X	X
0001	X	X	X	
0010	X	X		X
0011	X	X		
0100	X		X	X
0101	X		X	
0110	X			X
0111	X			
1000		X	X	X
1001		X	X	
1010		X		X
1011		X		
1100			X	X
1101			X	
1110				X
1111				

Table 193. PCI Configuration Space

Offset	Register Name	Description
0x00	pci_didvid	Device ID/Vendor ID
0x04	pci_srcr	Status Register/Control Register
0x08	pci_ccrid	Class Code/Revision ID
0x0C	pci_bhlc	BIST/Header Type/Latency Timer/Cache Line
0x10	pci_bar0	Base Address 0
0x14	pci_bar1	Base Address 1
0x18	pci_bar2	Base Address 2
0x1C	pci_bar3	Base Address 3
0x20	pci_bar4	Base Address 4
0x24	pci_bar5	Base Address 5
0x28	reserved	(Reserved)
0x2c	pci_sidsvid	Subsystem ID/Subsystem Vendor ID
0x30-38	reserved	(Reserved)
0x3C	pci_latint	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line
0x40	pci_rtotto	Defines retry timeout and trdy timeout parameters

**Table 194. Command Type for PCI Controller Configuration and Status Register Accesses**

Command Value pci_crp_ad_cbe[19:16]]	Command Type	Description
0x0	Read	Initiates a read of the PCI Controller Configuration and Status Register Accesses
0x1	Write	Initiates a write to the PCI Controller Configuration and Status Register Accesses
0x2 – 0xF	(Reserved)	Reserved for future use. Use of these values produce unpredictable results.

10.2.6 PCI Controller South AHB Transactions

The PCI Controller provides access to internal functionality within the IXP45X/IXP46X network processors. The PCI Controller provides access to the South AHB through the AHB Target Interface and the AHB Master Interface. The AHB Target Interface is used to accept transaction request from other AHB Masters. The AHB Master Interface is used to initiate transaction requests to other AHB Targets. The two DMA channels as well as the PCI Target Interface use the AHB Master Interface.

The AHB Target Interface can accept 8-bit (1 Byte) transactions, 16-bit transactions, and 32-bit transactions. Due to the South AHB not using byte enables, all 16-bit transactions to the PCI Controller AHB Target Interface must be implemented as consecutive-byte addresses. AHB protocol requires that 16-bit accesses be half-word address aligned, address bit 0 must be 0. Inability to do this will result in multiple byte wide transactions.

The AHB Master interface will initiate 8-bit (1 Byte) transactions and 32-bit (word) transactions only. The DMA engines will initiate only 32-bit transactions. PCI Target Interface initiated transactions will be 32-bit transactions. Sub 32-bit transactions — initiated by the PCI Target Interface — will be implemented as multiple 8-bit transactions initiated by the PCI Controller AHB Master on the AHB. For information on prioritization of the three functional blocks that use the PCI Controller AHB Master Interface, see next section that describes [“PCI Controller Functioning as Bus Initiator” on page 512](#).

10.2.7 PCI Controller Functioning as Bus Initiator

The IXP45X/IXP46X network processors can be used to initiate PCI transactions in one of three ways:

- Using the Non-Pre-fetch Registers — described in [“PCI Controller Configured as Host” on page 501](#)
The Non-Pre-fetch Registers allow various single 32-bit word PCI Cycles to be produced as well as 8 and 16 bit transfers. The Non-Pre-fetch Registers can be used to initiate Type 0 Configuration Cycles, Type 1 Configuration Cycles, Memory Cycles, I/O Cycles, and Special Cycles.
- Writing to the PCI Memory Cycle Address Space located between AHB address 0x48000000 and 0x4BFFFFFF as described in [“Initializing PCI Controller Configuration and Status Registers for Data Transactions” on page 505](#)
- Using the PCI Controller DMA channels — described in [“PCI Controller DMA” on page 536](#)

The remainder of the section shows example of each cycle type that may be initiated.



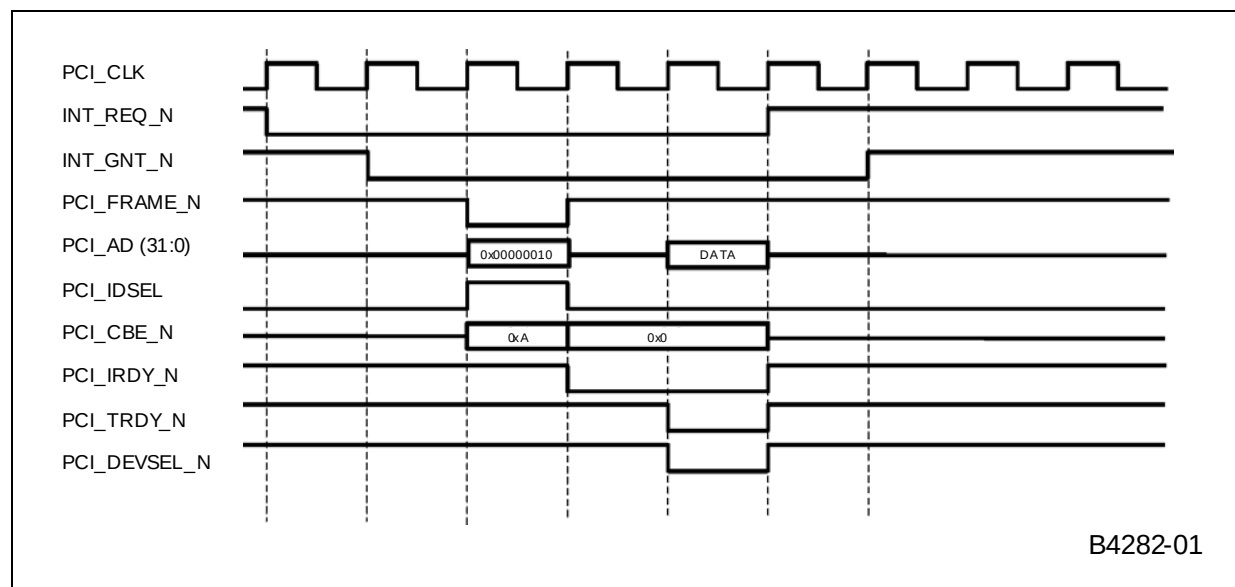
The details in this section are provided to understand some functional aspects of the PCI Controller on the IXP45X/IXP46X network processors. For complete details please refer to the *PCI Local Bus Specification*, Rev. 2.2.

10.2.7.1 Initiated Type-0 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The Configuration transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function 0, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It may be driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xA, written on the PCI_CBE_N bus during the PCI Bus address phase, signifies that this is a PCI Bus Configuration Read Cycle.

Figure 80. Initiated PCI Type-0 Configuration Read Cycle



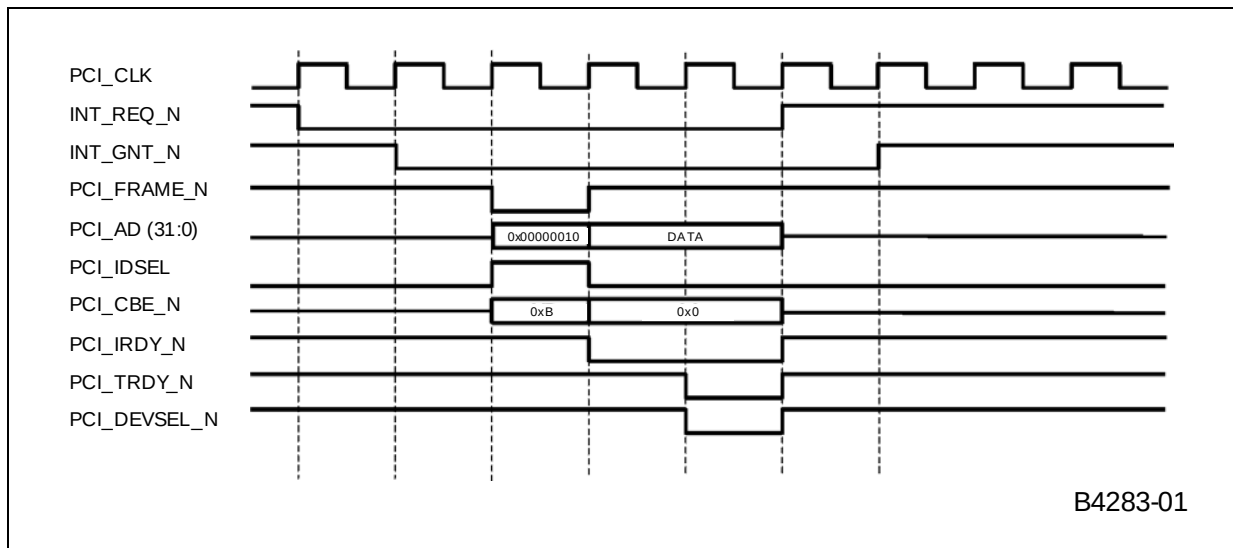
10.2.7.2 Initiated Type-0 Write Transaction

The following transaction is a PCI Configuration Write Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function number 2, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It may be driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xB written on the PCI_CBE_N bus — during the address phase — signifies that this is a PCI Bus Configuration Write Cycle.



Figure 81. Initiated PCI Type-0 Configuration Write Cycle

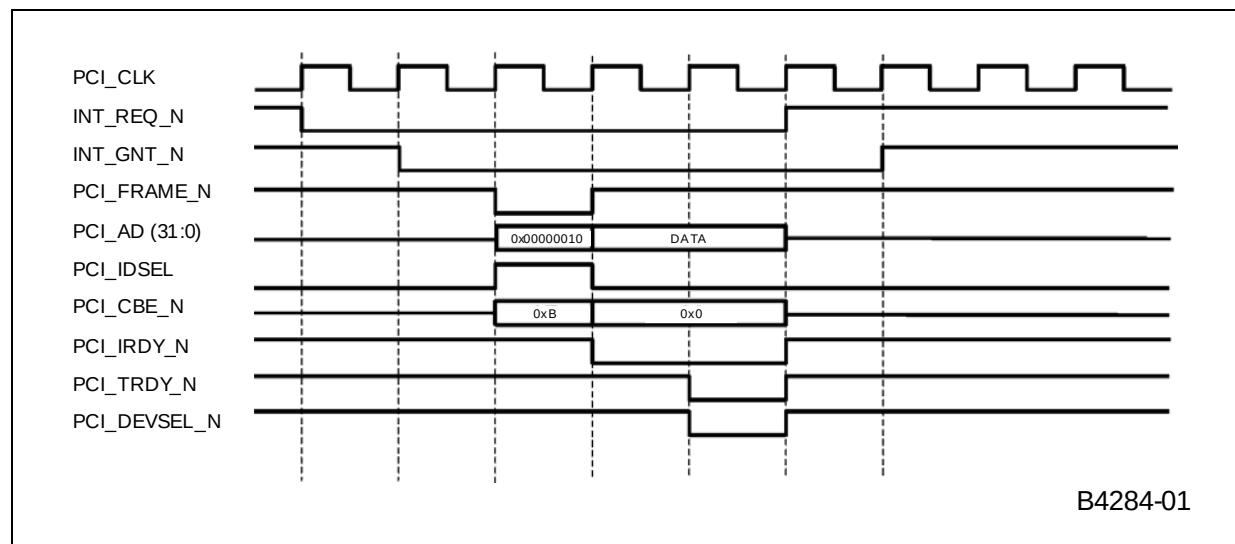


10.2.7.3 Initiated Type-1 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to PCI bus segment 0, Device number 0, Function 0, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01 being located in bits 1:0 of the PCI_AD bus during the address phase denotes a Type 1 PCI Configuration cycle.

A hexadecimal value of 0xA — written on the PCI_CBE_N bus during the address phase — signifies that this is a PCI Bus Configuration Read Cycle. Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore may be extended by several clocks and is not shown here.

**Figure 82. Initiated PCI Type-1 Configuration Read Cycle**

10.2.7.4 Initiated Type-1 Write Transaction

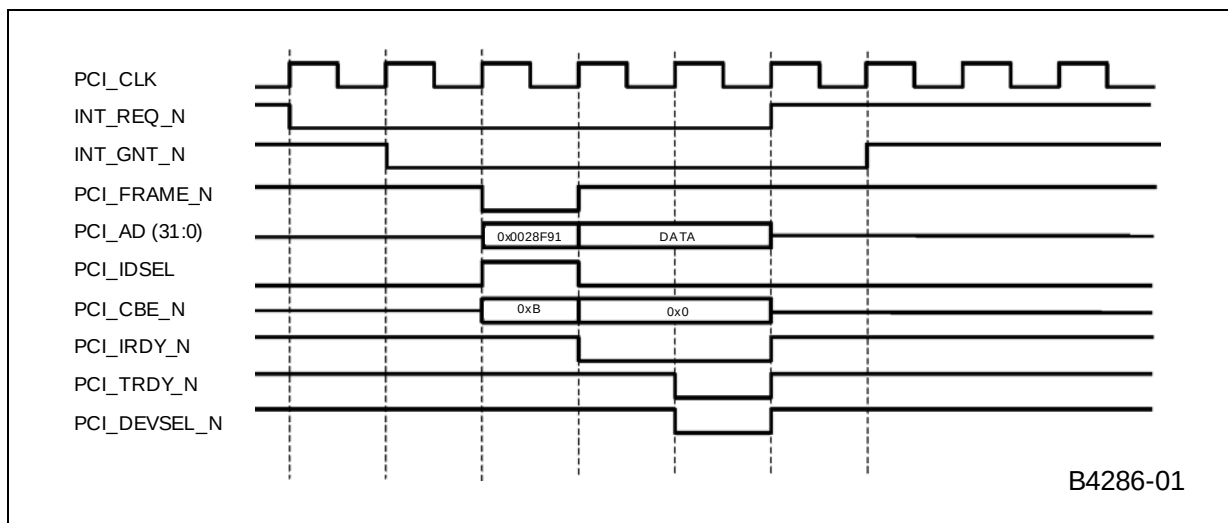
The following transaction is a PCI Configuration Write working-site cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to PCI bus segment 5, Device number 3, Function number 7, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01 — being located in bits 1:0 of the PCI_AD bus during the address phase — denotes a Type 1 PCI Configuration cycle. A hexadecimal value of 0xB, written on the PCI_CBE_N bus during the address phase, signifies that this is a PCI Bus Configuration Write Cycle.

Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore may be extended by several clocks and is not shown here.



Figure 83. Initiated PCI Type-1 Configuration Write Cycle

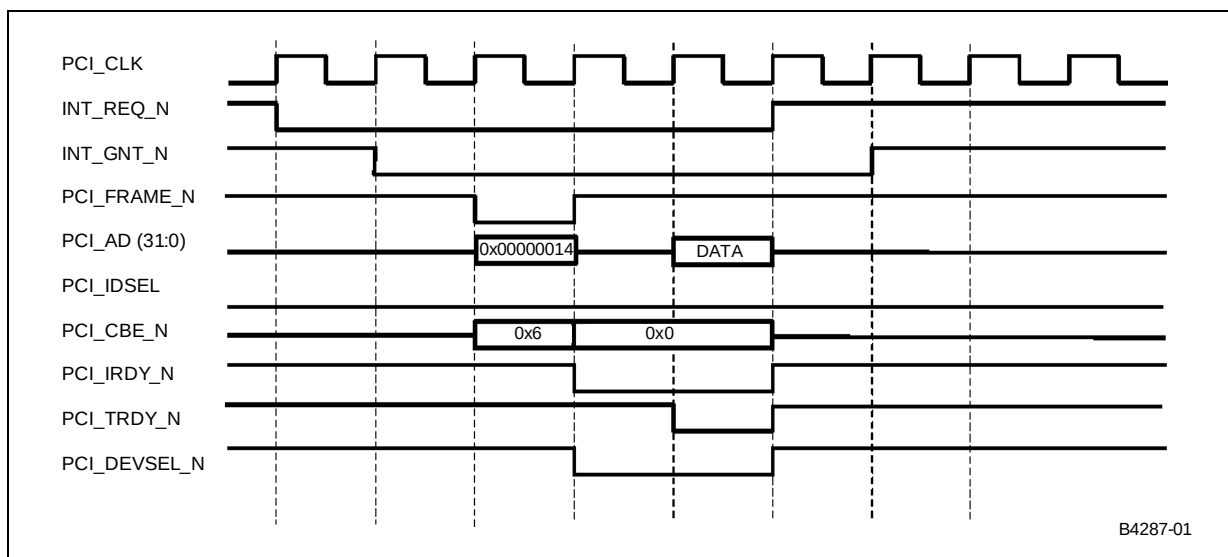


10.2.7.5 Initiated Memory Read Transaction

The following transaction is a PCI Memory Read Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type.

A hexadecimal value of 0x6, written on the PCI_CBE_N bus during the address phase, signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Figure 84. Initiated PCI Memory Read Cycle



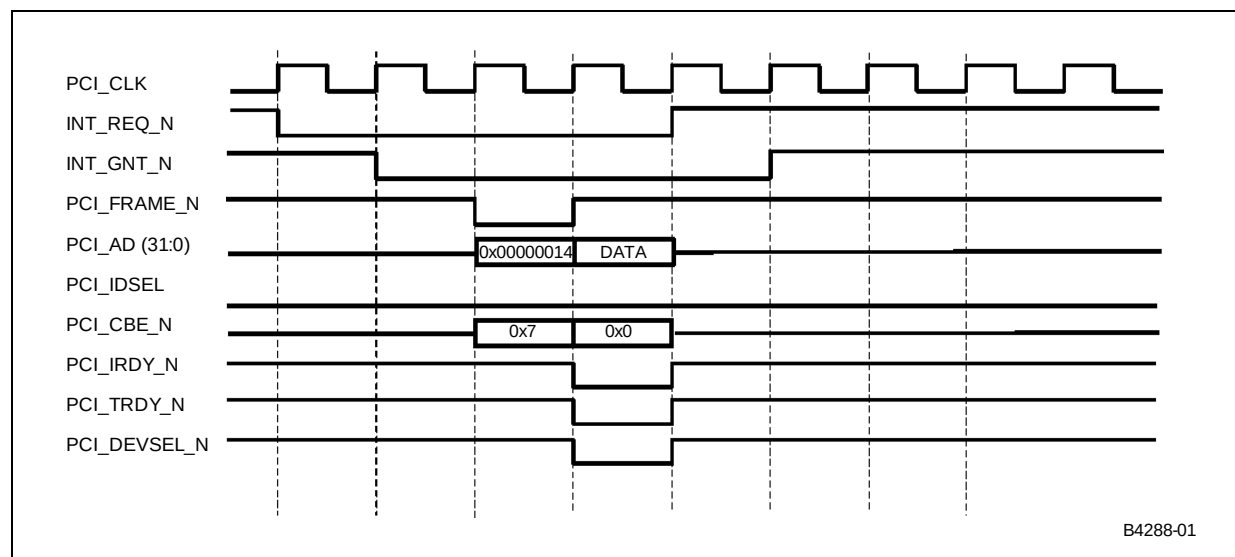


10.2.7.6 Initiated Memory Write Transaction

The following transaction is a PCI Memory Write Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear-increment transfer type. A hexadecimal value of 0x7 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Notice that on this transaction the PCI_DEVSEL_N signal timing is different. This signal-timing differential is due to the fact that the PCI_DEVSEL_N signal must become active within the first three clocks after the PCI_FRAME_N becoming active. This requirement could be different for every device that is on the PCI Bus. There is also no relationship to when PCI_TRDY_N becomes active other than the PCI_TRDY_N signal must not become active prior to the PCI_DEVSEL_N signal becoming active.

Figure 85. Initiated PCI Memory Write Cycle

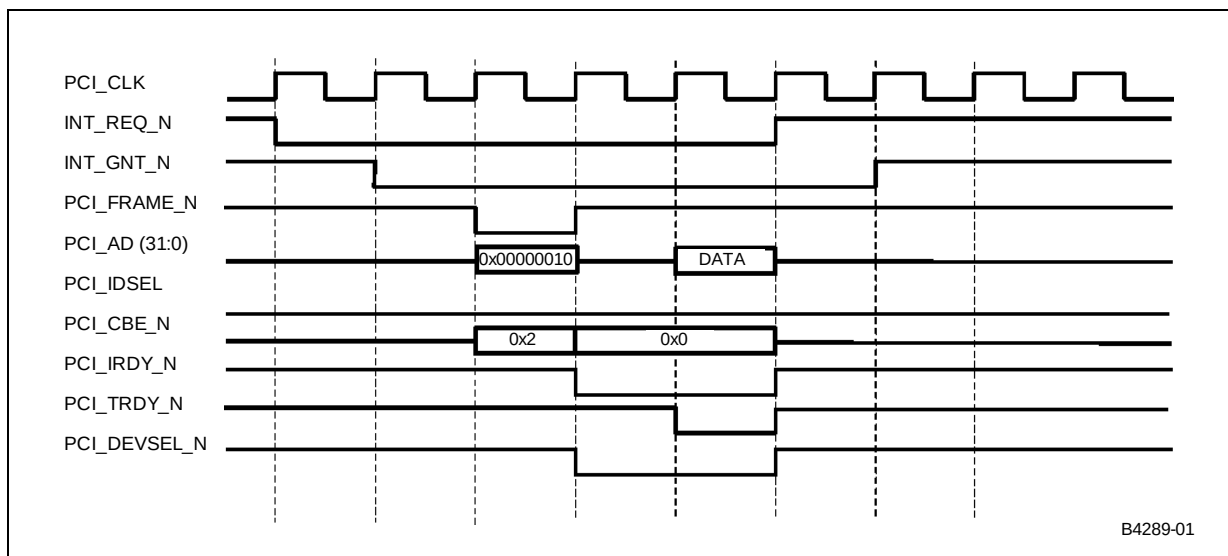


10.2.7.7 Initiated I/O Read Transaction

The following transaction is a PCI I/O Read Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to address location hexadecimal 0x00000010.

A hexadecimal value of 0x2 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus I/O Read Cycle.

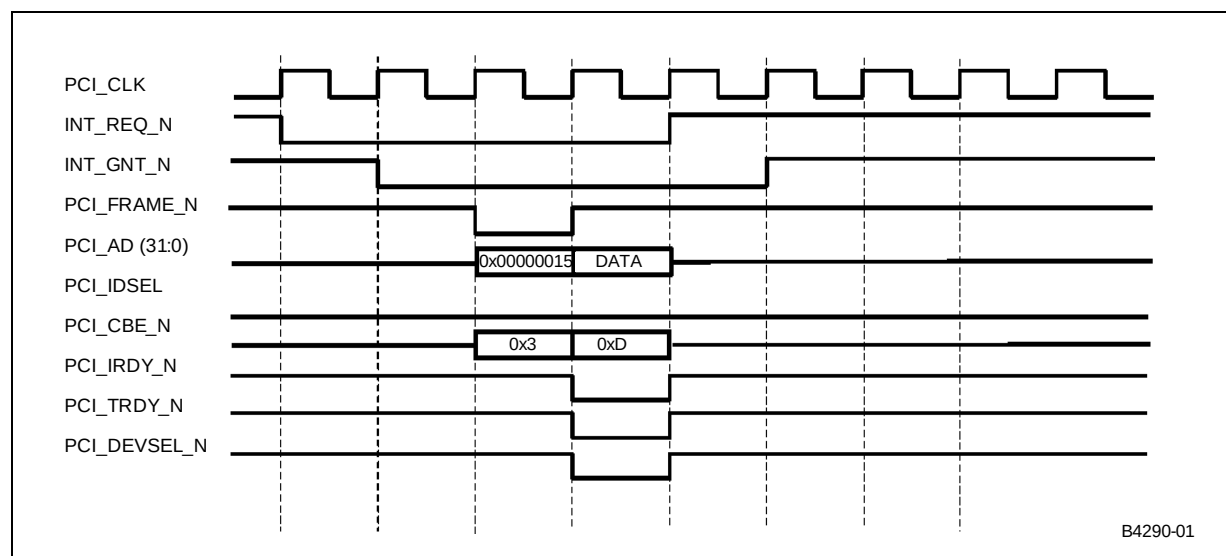
Figure 86. Initiated PCI I/O Read Cycle



10.2.7.8 Initiated I/O Write Transaction

The following transaction is a PCI I/O Write Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to address location hexadecimal 0x00000015. The value of binary 01 in PCI_AD (1:0) indicates that the transfer is a valid byte address of the first byte of 32-bit word address 0x00000014 ($0x00000014 + 0x00000001 = 0x00000015$).

The byte-enables being 0xD — during the data transfer — signify that the transfer is a byte transfer to the above-mentioned address. A hexadecimal value of 0x3 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus I/O Write Cycle.

**Figure 87. Initiated PCI I/O Write Cycle**

10.2.7.9 Initiated Burst Memory Read Transaction

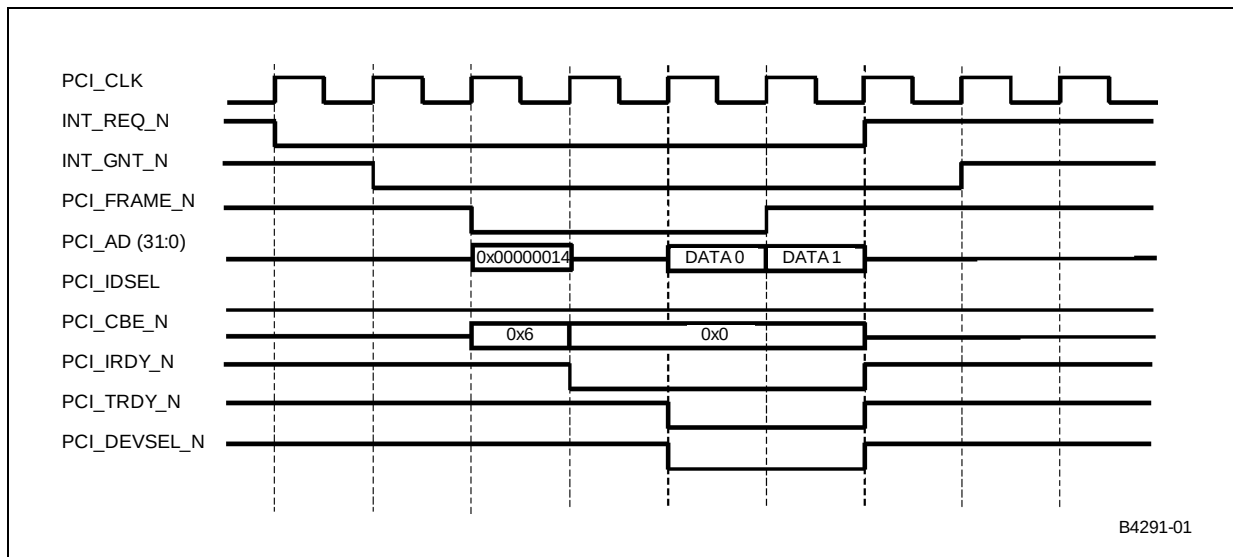
The following transaction is a two word bursting PCI Memory Read Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to initial address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type. The second data word transferred will be from address hexadecimal 0x00000018.

A hexadecimal value of 0x6 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Read Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP45X/IXP46X network processors.



Figure 88. Initiated PCI Burst Memory Read Cycle



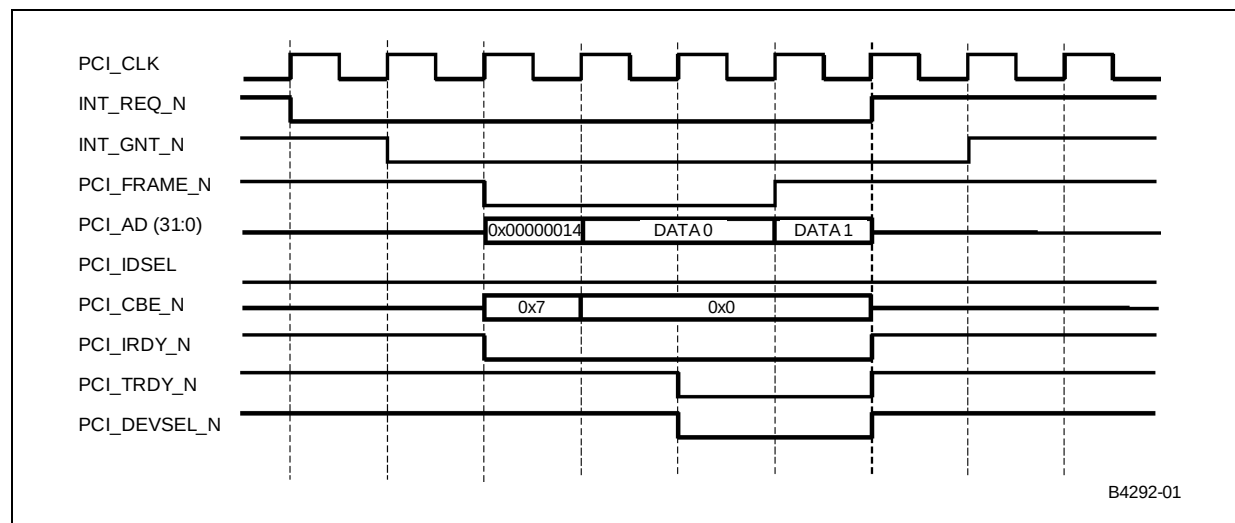
10.2.7.10 Initiated Burst Memory Write Transaction

The following transaction is a two word bursting PCI Memory Write Cycle initiated from the IXP45X/IXP46X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP45X/IXP46X network processors. The transaction is initiated to initial address location hexadecimal 0x00000014.

The value of binary 00 in PCI_AD (1:0) indicates that this is an linear increment transfer type. The second data word transferred will be from address hexadecimal 0x00000018.

A hexadecimal value of 0x7 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Write Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP45X/IXP46X network processors.

**Figure 89. Initiated PCI Burst Memory Write Cycle**

10.2.8 PCI Controller Functioning as Bus Target

The IXP45X/IXP46X network processors can be the target of PCI transactions. Operating as a PCI target, the PCI bus can accept Memory Cycles, I/O Cycles, or Configuration Cycles. Target transactions can take place independent of the Host/Option configuration of the IXP45X/IXP46X network processors. Please refer to [Table 189, “PCI Target Interface Supported Commands” on page 498](#) for additional information on supported commands.

Only Type 0 Configuration Cycles are supported.

Timing diagrams are not shown for the target transactions because they are similar to initiated transactions. The only differences are the PCI devices that source/sink the various PCI signals.

For target-read transactions, a retry will be issued upon the receipt of a request to transfer data. Between the time that the retry occurs and the access to the IXP45X/IXP46X network processors reoccurs, the PCI Controller retrieves the data from the previously requested location.

For additional details, see the *PCI Local Bus Specification*, Rev. 2.2.

10.2.9 PCI Controller Door Bell Register

The PCI Controller has two registers that make up the Door Bell register logic on the IXP45X/IXP46X network processors. These two registers are the AHB Door Bell (PCI_AHBDOORBELL) register and the PCI Door Bell Register (PCI_PCIDDOORBELL).

An external PCI device writes the AHB Door Bell (PCI_AHBDOORBELL) register to generate an interrupt signal to the Intel XScale processor. If the AHB doorbell interrupt is enabled (PCI_INTEN.ADBEN = 1) in the PCI interrupt registers, any bit set to logic 1 in the AHB Door Bell (PCI_AHBDOORBELL) will force the interrupt signal to occur.

The AHB Door Bell (PCI_AHBDOORBELL) register is set from the PCI bus only by writing logic 1 to the register. Writing logic 1 to the set bits from the SOUTH AHB clears bits that are set in the AHB Door Bell (PCI_AHBDOORBELL).

An example of using the AHB Door Bell (PCI_AHBDOORBELL) is as follows:



1. An external PCI device writes logic 1 to a bit or pattern of bits to generate an interrupt to the Intel XScale processor.
2. The AHB agent reads the AHB Door Bell (PCI_AHBDOORBELL) register and writes logic 1(s) to all set bits clear the set bit(s). This in turn de-assert the interrupt generated to the Intel XScale processor.

Using the South AHB, the Intel XScale processor writes the PCI Door Bell Register (PCI_PCIDOOORBELL) to generate an interrupt to an external PCI device over the PCI_INTA_N signal. Any bit set to logic 1 in the PCI Door Bell Register (PCI_PCIDOOORBELL) will generate the PCI interrupt if the PCI doorbell interrupt is enabled the PCI Interrupt Enable Register (PCI_INTEN).

The PCI Door Bell Register (PCI_PCIDOOORBELL) register can only be written by the AHB. The external PCI device must write logic 1 to all set bits in the PCI Door Bell Register (PCI_PCIDOOORBELL) in order to clear the bits set by the Intel XScale processor.

An example of using the PCI Door Bell (PCI_PCIDOOORBELL) is as follows:

1. The Intel XScale processor writes logic 1 to a bit or pattern of bits in the PCI Door Bell Register (PCI_PCIDOOORBELL) to generate an interrupt on the PCI bus using PCI_INTA_N.
2. An external PCI device reads the PCI Door Bell Register (PCI_PCIDOOORBELL) and writes logic 1(s) to all set bits to clear the set bit(s). This causes the interrupt that is asserted to de-assert.

10.3 Functional Description

10.3.1 PCI Byte-Enable Generation

The byte enables for single PCI transactions are generated based on the type of AHB access (direct read/write or non-prefetch read/write), address, transfer size (8-bit, 16-bit, 32-bit), and the settings in effect for AHB endianness and data swapping modes. All 32-bit accesses obviously assert all byte enables when the transaction is sent to the PCI Initiator. Since the AHB Slave Interface only supports bursts with a size of 32-bits, PCI burst transactions will always have all byte enables asserted. [Table 198](#) shows the byte enables for 8-bit and 16-bit single PCI read and write cycles generated by direct access of the AHB Slave Interface. PCI cycles generated by the non-prefetch CSRs use the value of the NP_BE field of the pci_np_cbe CSR for the byte enables.

10.3.2 PCI Core

This block generates the PCI compliant protocol logic. It operates either as an initiator or a target device on the PCI Bus. As an initiator, all bus cycles are generated by the core. As a PCI target, the core responds to bus cycles that have been directed towards it.

On the PCI Bus, the core supports interrupts, 32 bit data path, 32 bit addressing, and a single configuration space. The local configuration registers (CSRs) are accessible from the PCI Bus or from the Intel XScale® Processor through the AHB bus.

There are four 8 word deep data FIFOs and a 4 word deep address FIFO in the core. The separate slave and master data FIFOs allow simultaneous operations and multiple outstanding PCI bus transfers. The initiator address FIFO can accumulate up to four addresses which can be PCI reads or writes.



10.3.2.1 PCI Target Interface

The PCI Target Interface provides read/write access to AHB agents, PCI Controller PCI Configuration registers (through Configuration cycles), and some PCI Controller CSRs. The interface performs the PCI target actions in response to external PCI Master transactions. Table 195 lists the supported command types.

The interface does not support the following features:

- Lock cycles
- VGA palette snoop
- Dual address cycles
- Cache-line, wrap-mode addressing
- Configuration Type 1 read/write cycles

Table 195. PCI Target Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	not supported
0x1	Special Cycle	not supported
0x2	I/O Read	supported
0x3	I/O Write	supported
0x4	Reserved	
0x5	Reserved	
0x6	Memory Read	supported
0x7	Memory Write	supported
0x8	Reserved	
0x9	Reserved	
0xa	Configuration Read	supported
0xb	Configuration Write	supported
0xc	Memory Read Multiple	Converted to Memory Read
0xd	Dual Address Cycle	not supported
0xe	Memory Read Line	Converted to Memory Read
0xf	Memory Write and Invalidate	converted to Memory Write

10.3.2.1.1 PCI Bus Access to PCI Configuration Registers

An access to PCI Configuration registers occurs when the PCI_IDSEL input is asserted, the PCI command is a configuration read or write, and PCI_AD[1:0] = 00 indicating a type 0 configuration cycle. The selected register is indicated by PCI_AD[7:2]. Accesses are single-word only, the Target Interface will disconnect any burst longer than 1 word. During reads, byte enables are ignored and the full 32-bit register value is returned. Read accesses to unimplemented registers complete normally on the bus and return all zeroes. During writes, the PCI byte enables determine the byte(s) to be written within the addressed register. Write accesses to unimplemented registers complete normally on the bus but the data is discarded.



10.3.2.1.2 PCI Bus Access to PCI Controller CSRs

PCI Controller CSRs are accessed from PCI through read or write transactions whose address matches the PCI base address register `pci_bar4`. `Pci_bar4` is written by the PCI Host during the bus configuration process to map the PCI Controller CSRs into the external PCI bus memory map. Currently, only two of the PCI Controller CSRs are write accessible from the PCI bus when not in PCI test mode (`exp_pcitest = 1`): `pci_pcidoorbell` and `pci_ahbdoorbell`. When `exp_pcitest = 0`, all CSRs are write accessible from the PCI bus. All registers can be read from PCI.

10.3.2.1.3 PCI Bus Access to AHB Address Space

Internal AHB address space is accessed from the PCI bus through read or write transactions whose address matches one of the implemented PCI base address registers `pci_bar0,1,2,3`, or `5`. These registers are written by the PCI Host during the bus configuration process to map five AHB address regions into the external PCI bus address map.

10.3.2.1.4 PCI Target Write Accesses

A PCI target memory write occurs if the PCI address matches one of the PCI base address registers `pci_bar0,1,2,3` and the PCI command is Memory Write or Memory Write and Invalidate. The address, data, byte enables, and BAR identifier (indicating which PCI base address register was hit) are written to the Target Receive FIFO. If the FIFO fills up during the transfer, the PCI Controller signals a target disconnect to the external master. If the FIFO is full at the start of a write access, a retry is signaled on the bus. All combinations of data phase byte enables are supported.

A PCI target I/O write occurs if the PCI address matches the PCI base address register `pci_bar4` and the PCI command is a I/O Write. In this case, the PCI Target Interface disconnects the transfer after the 1st data phase.

10.3.2.1.5 PCI Target Read Accesses

A PCI target memory read occurs if the PCI address matches one of the PCI base address registers and the PCI command is Memory Read, Memory Read Line, or Memory Read Multiple. The address and BAR identifier (indicating which PCI base address register was hit) are written to the Target Receive FIFO and the transaction is retried on the PCI bus. After AHB read data is written into the Target Transmit FIFO by the AHB Master Interface, the PCI Target Interface delivers the data to the initiator when the initiator retries the transfer. If the Target Transmit FIFO becomes empty, the PCI Target Interface issues a target disconnect to the initiator until it becomes non-empty again. The interface ignores byte enables for PCI memory reads and delivers all addressed words to the initiator.

A PCI target I/O read occurs if the PCI address matches the PCI base address register `pci_bar4` and the PCI command is an I/O Read. In this case, the PCI Target Interface disconnects the transfer after the 1st data phase.

When the read data is returned into the Target Transmit FIFO, the PCI Controller begins to decrement a discard timer. If the external PCI bus initiator has not repeated the read by the time the timer reaches zero, the PCI Controller discards the read data and invalidates the delayed read address. The discard timer counts 2^{15} (32768) PCI clocks. Once the timer expires, the next PCI target read is treated as a new operation.

10.3.2.2 PCI Initiator Interface

The Master Interface provides bus mastering capability in response to PCI requests received from an AHB master or from the PCI Controller DMA channels. Requests are buffered in the Initiator Request FIFO and handled by the PCI Core Initiator Interface.



This interface receives the address, word count, byte enables and PCI command type from the AHB side and performs the specified transaction on the PCI bus, handling all bus protocol and retry/disconnect situations. [Table 196](#) lists the supported transaction types.

Table 196. PCI Initiator Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	supported
0x1	Special Cycle	supported
0x2	I/O Read	supported
0x3	I/O Write	supported
0x4	Reserved	
0x5	Reserved	
0x6	Memory Read	supported
0x7	Memory Write	supported
0x8	Reserved	
0x9	Reserved	
0xa	Configuration Read	supported
0xb	Configuration Write	supported
0xc	Memory Read Multiple	supported
0xd	Dual Address Cycle	not supported
0xe	Memory Read Line	supported
0xf	Memory Write and Invalidate	not supported

10.3.2.2.1 Initiator Write Transactions

The PCI Master Interface receives write requests from the AHB Slave Interface or AHB-to-PCI DMA Controller via the Initiator Request FIFO. Write data is supplied using the Initiator Transmit FIFO. The Master Interface transfers the indicated number of words from the FIFO to the PCI bus. The following rules apply to the write transfers:

- If the Transmit FIFO becomes empty, the Master Interface terminates the cycle on the PCI bus. When more data becomes available, the transfer resumes using the address of the next word to be delivered.
- If the master's Latency Timer expires, the Master Interface terminates the cycle on the PCI bus (see [“Master Latency Timer”](#) on page 526 for a description of the Latency Timer). The transfer resumes at the first opportunity using the address of the next word to be delivered.
- If a retry or target disconnect is received before the transfer ends, the Master Interface resumes the transfer at the first opportunity using the address of the next data word to be delivered.
- If a master abort occurs, all of the write data is discarded and the Received Master Abort bit is set in the PCI Configuration status register.
- If a target abort is received, all of the write data is discarded and the Received Target Abort bit is set in the PCI Configuration status register.



10.3.2.2.2 Initiator Read Transactions

The PCI Master Interface receives read requests from the AHB Slave Interface or PCI-to-AHB DMA Controller via the Initiator Request FIFO. Read data is supplied to the AHB side using the Initiator Receive FIFO. The Master Interface transfers the indicated number of words from the PCI bus to the FIFO. The following rules apply to the read transfers:

- For memory read operations, if the cache line size indicated in the PCI Configuration Register `pci_bhlc` is zero, the Memory Read command is used on PCI. For non-zero cache-line sizes, the starting address and word count of the transfer are used to determine how many cache lines will be read. If one full cache line is to be read, the Memory Read Line command is used. If two or more full cache lines, the Memory Read Multiple command is used. Note that read commands may change from transaction to transaction during long burst reads that are broken up by retries or disconnects.
- If the Receive FIFO becomes full, the Master Interface terminates the cycle on the PCI bus using a disconnect. When room becomes available, the transfer resumes using the address of the next word to be read.
- If the master's Latency Timer expires, the Master Interface terminates the cycle on the PCI bus. The transfer resumes at the first opportunity using address of the next word to be read.
- If a retry or target disconnect is received before the transfer ends, the Master Interface resumes the transfer at the first opportunity using the address of the next data word to be read.
- If a master abort occurs, the AHB side transaction is terminated with an ERROR response and the Received Master Abort bit is set in the PCI Configuration status register.
- If a target abort is received, the AHB side transaction is terminated with an ERROR response and the Received Target Abort bit is set in the PCI Configuration status register.

10.3.2.2.3 Master Latency Timer

When the Initiator Interface begins PCI transaction it begins to decrement its master latency timer. When the timer value reaches zero, the PCI grant input is checked. If the grant is deasserted, the Initiator Interface deasserts `PCI_FRAME_N` (if it is still asserted) at the earliest opportunity (normally the next data phase for all transactions). The transfer is resumed as soon as possible at the address of the next word.

10.3.2.2.4 Initiator Special Cycles

Special Cycles are broadcast to all PCI targets on the bus and are terminated by Master Abort. No error is signaled in this case.

10.3.2.3 PCI Host Functions

A logic high state on the `exp_pcihost` input port indicates that the PCI Controller should function in host mode. In this mode, access to the PCI Configuration registers is only allowed from the AHB via the CSR-based configuration access port. In non-host mode (`exp_pcihost = 0`), access is only allowed from the external PCI bus.

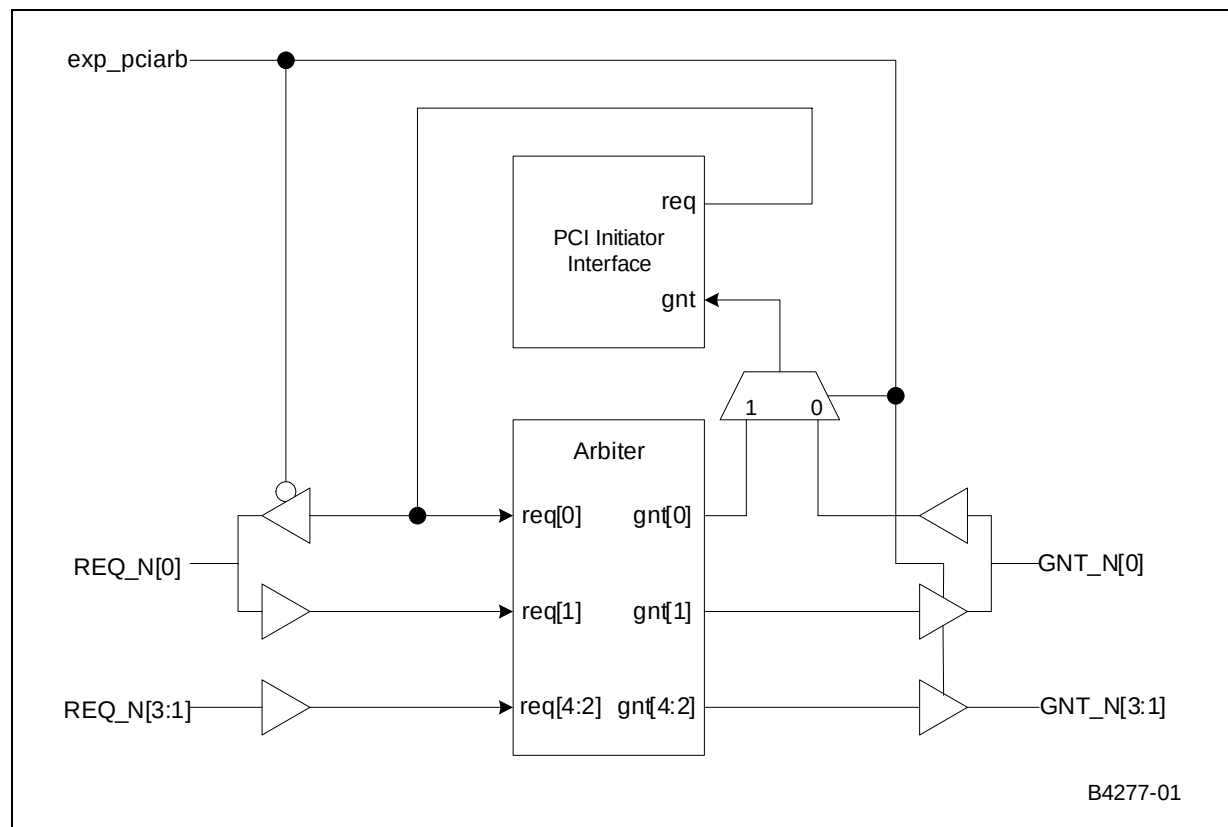
The following sections describe PCI host functions that do not directly affect PCI Controller functionality.



10.3.2.3.1 PCI Arbiter

The PCI Controller contains a PCI bus arbiter that supports four external masters in addition to the PCI Controller's Initiator Interface. To enable the arbiter, the `exp_pciarb` pin must be a logic 1. When the arbiter is enabled, the PCI Controller PCI request and grant signals are routed internal to the chip and cannot be viewed externally. Figure 90 shows a conceptual view of the arbiter request and grant signal wiring. The PCI Master Interface, Arbiter, and 2:1 multiplexer are part of the PCI Controller.

Figure 90. PCI Controller Arbiter Configuration



The arbiter uses a simple round-robin priority algorithm. The arbiter asserts the grant signal corresponding to the next request in the round-robin during the current executing transaction on the PCI bus (i.e. arbitration is hidden). If the arbiter detects that an initiator has failed to assert the `PCI_FRAME_N` signal after 16 cycles of both grant assertion and a PCI bus idle condition, the arbiter deasserts the grant. That master does not receive any more grants until it deasserts its request for at least one PCI clock cycle. Bus parking is implemented in that the last bus grant will stay asserted if no request is pending. To prevent bus contention, if the PCI bus is idle, the arbiter never asserts one grant signal in the same PCI cycle in which it deasserts another. It deasserts one grant, then asserts the next grant after one full PCI clock cycle has elapsed to provide for bus driver turnaround.

10.3.2.3.2 PCI Interrupt Inputs

If used as a PCI host, up to 4 PCI interrupt signals can be wired as GPIO inputs to the GPIO Controller function (not a part of the PCI Controller) and further presented to the Interrupt Controller function (again, not a part of the PCI Controller) to generate an



interrupt to the Intel XScale processor. The PCI Controller provides no hardware support for these interrupts. By specification, PCI interrupts are level-sensitive and asserted/deasserted asynchronously to the PCI clock.

10.3.2.4 PCI Controller Clock and Reset Generation

The PCI Reset and PCI clock signals can be provided using general-purpose input/output (GPIO) outputs or from an external source. The GPIO block is not part of the PCI controller. See below for particular signals which may be used on various products for these functions:

Table 197. PCI CLOCK and RESET Sourcing

Function	Intel® IXP42X	IXP45X / IXP46X network processors
Clock Source	GPIO14	GPIO14
Reset Source	GPIO[13:0]	GPIO[13:0]

Both signals can be sourced from an external device as well. The Intel XScale processor can generate the PCI reset and PCI clock outputs to satisfy the reset timing requirements of the PCI bus.

A PCI startup sequence could be as follows:

1. Power-on reset occurs to the IXP45X/IXP46X network processors, the Intel XScale processor starts execution (internal PLL assumed locked and internal clocks stable).
2. Software configures PCI reset and PCI clock GPIOs as outputs driving 0. A pull-down on the GPIO pin chosen to drive the PCI reset signal is required. This pull-down is required because the GPIO are at a tri-stated value until the device comes completely out of reset and the PCI reset needs to be low from the start.
3. Wait 1ms to satisfy minimum reset assertion time of the PCI specification.
4. Configure the PCI clock GPIO for the proper PCI bus frequency (defined in the section GPIO).
5. Enable the PCI clock GPIO to drive the PCI clock
6. Wait 100 μ s to satisfy the "minimum reset assertion time from clock stable" requirement of the PCI specification.
7. Set the PCI reset GPIO output to drive a 1. This releases the PCI bus.

Note: The PCI reset can be asserted and de-asserted asynchronously with respect to the PCI clock. It is also important to note the PCI reset signal cannot be the same signal as the RESET_IN_N signal going to the processor due to PCI reset timing and PCI initialization requirements.

10.3.2.5 PCI Configuration Register Access

As a PCI host performing configuration of the bus, the Intel XScale processor must have read/write access to all of the PCI Configuration registers in the PCI Controller. This access is provided via a set of three CSRs accessible from the AHB bus as described in ["AHB Accesses of Local PCI Configuration Registers" on page 534](#).

10.3.2.5.1 PCI Configuration

Configuration of the PCI bus is performed by a PCI agent using configuration read and write cycles. All devices on the bus have their configuration space read by the agent to determine the attributes of the device. The agent then writes the appropriate configuration registers in each device to, for example, set up the PCI memory map, and



enable/disable certain bus signalling characteristics depending on the capabilities of the devices on the bus. The PCI Controller supports bus configuration both as a host performing configuration transactions, and as a non-host responding to configuration transactions from an external agent.

In non-host mode, select registers must be initialized by the Intel XScale processor before the PCI Controller can respond to PCI configuration cycles from an external agent. The Subsystem ID and Subsystem Vendor ID are two such registers. The Initialization Complete bit in the control and status register, when cleared, forces a retry on the PCI bus in response to configuration cycles. This bit is cleared at reset thus holding off configuration by an external agent until the PCI Controller configuration registers are properly initialized. When the Intel XScale processor has completed its initialization of the necessary registers, it sets the bit to allow PCI configuration to proceed. A special test mode, indicated by a logic 0 level on the exp_pcitest input port, overrides this process and tells the PCI Controller to respond to PCI configuration cycles regardless of the state of Initialization Complete.

When the Intel XScale processor is the PCI host performing system configuration, it is responsible for setting up the PCI Controller configuration registers as well as those of all the other devices on the bus. To support this, access to local PCI configuration registers is provided via a CSR-based configuration access port as described in [“AHB Accesses of Local PCI Configuration Registers” on page 534](#). Likewise, configuration read/write cycles can be generated on the PCI bus using a CSR-based PCI access port as described in [“AHB Non-Prefetch PCI Accesses” on page 535](#).

10.3.2.6 PCI Pad Drive Strength Compensation Support

The PCI Core supports PCI pad strength compensation via the exp_rcomp_complete input. When exp_rcomp_complete is at a logic 0 state, the PCI Core Initiator and Target interfaces will be inhibited from bus operations thus preventing any switching activity on the PCI bus from this device. In addition, the PCI arbiter grant outputs pcc_gnt_n[3:0] will remain in the de-asserted state regardless of the state of the request inputs. This feature allows an external drive strength compensation circuit to inhibit any switching of the PCI output drivers until an optimum drive for the PCI pads can be determined based on temperature and voltage levels on the die. When this drive is determined, exp_rcomp_complete is asserted to a logic 1 state to allow normal PCI functioning of the PCI Core.

10.3.2.6.1 Effect on Initiator Interface

When exp_rcomp_complete is logic 0, the PCI Controller Initiator interface is inhibited from starting a PCI master cycle that may be queued in the Initiator Request FIFO. The PCI bus request is not asserted. When exp_rcomp_complete is a logic 1, normal operation of the Initiator Interface is enabled. If one or more requests is waiting in the Request FIFO when exp_rcomp_complete changes state from 0 to 1, those requests will be processed in a normal manner.

10.3.2.6.2 Effect on Target Interface

When exp_rcomp_complete is logic 0, the PCI Controller Target interface is inhibited from responding to PCI cycles initiated from PCI masters. The interface will not drive any PCI signals and the cycle will terminate with a Master Abort. When exp_rcomp_complete asserts to a logic 1, the Target Interface will respond normally to the next PCI cycle that targets this device, usually a Configuration Read.



10.3.2.6.3 Effect on Internal PCI Arbiter

If the internal PCI arbiter is enabled, a low level on `exp_rcomp_complete` will block all requests from external PCI masters and cause the arbiter to park the bus on the local PCI Initiator Interface. This has the effect of preventing any switching on the arbiter grant outputs. When `exp_rcomp_complete` is a logic 1, the requests from external masters are enabled and the arbiter functions normally.

10.3.2.7 AHB Master Interface

The AHB Master Interface provides read/write access to AHB slaves and local CSRs for transactions generated by external PCI Initiators. It also services read/write requests from the DMA Controller channels to effect DMA transfers between the AHB and PCI busses. Both big-endian and little-endian addressing are implemented depending on the state of the `pci_csr.ABE` bit. Three virtual AHB masters can initiate transfers using the AHB Master Interface:

1. Transfers originating from an external PCI master are presented to the AHB Master Interface by the PCI Target Interface via the Target FIFOs in the PCI Core. The AHB Master Interface reads the PCI address and control information from the Target Receive FIFO and translates the PCI transaction to an appropriate AHB transaction. For writes, data is written to the Target Receive FIFO by the PCI Core and read out to the AHB bus by the AHB Master Interface. During reads, the AHB Master Interface performs an appropriate read transaction and writes the data to the Target Transmit FIFO.
2. The AHB-to-PCI DMA channel uses the AHB Master Interface to read data from an AHB agent and send a PCI write request to the PCI Core via the Initiator Request and Initiator Transmit FIFOs.
3. The PCI-to-AHB DMA channel uses the AHB Master Interface to read previously requested PCI read data from the Initiator Receive FIFO and write this data to an AHB agent.

Arbitration for control of AHB mastering resources is carried out on two levels. On the first level, PCI requests and DMA requests alternate for priority access. On the second level, the two DMA channels alternate for priority access. This arbitration scheme balances the high bandwidth DMA traffic with the lower bandwidth PCI traffic.

10.3.2.7.1 PCI Address Translation to AHB Address

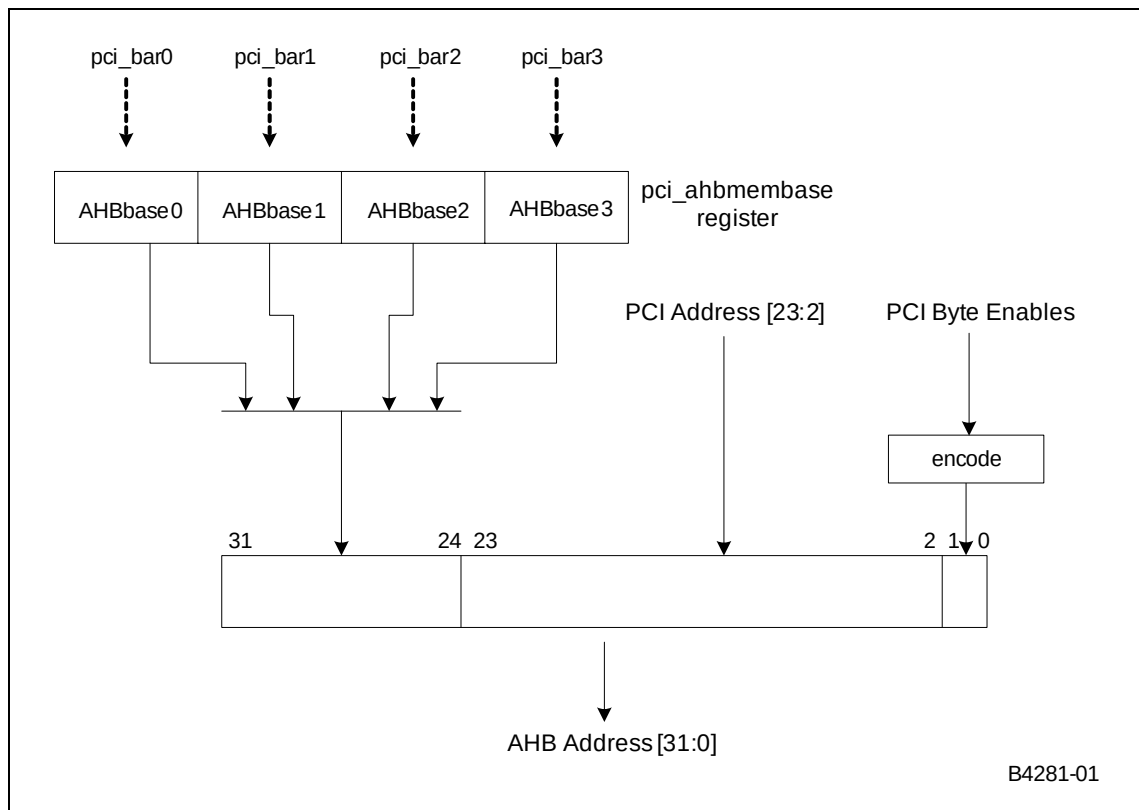
The PCI address received from the PCI Core is translated to an AHB address as follows:

- The AHB Master Interface reads a BAR indicator (BAR ID) from the Core to determine which of the 6 PCI Configuration BARs was targeted.
- If one of the lower four BARs was targeted (`PCI_BAR0/1/2/3`), the bits 23:2 of the PCI address map to bits 23:2 of the AHB address bits and an 8-bit field in the `PCI_AHBMEMBASE` register provides the upper 8 address bits of the AHB address. In this manner, each 16MB address range on the PCI bus can be mapped to any 16MB region in the full 4GB AHB address space by appropriately initializing the four base address fields in `pci_ahbmembase`. Bits 1:0 of the AHB address are derived from the PCI byte enables. [Figure 91](#) shows the address translation mechanism.
- If `PCI_BAR4` was targeted, the access is directed to the PCI accessible CSRs in the PCI Controller and no AHB transaction is performed.
- If `PCI_BAR5` was targeted, bits 7:2 of the PCI address map to bits 7:2 of the AHB address bits and the 24-bit IOBase field of the `PCI_AHBIOWBASE` register provides the upper 24 bits of the AHB address. Bits 1:0 of the AHB address are derived from the PCI byte enables.



Since the PCI controller performs pre-fetches when doing reads to the PCI_BAR0/1/2/3 address windows, the PCI_AHBMEMBASE must not be programmed to access any AHB I/O space or the AHB Queue Manager. To access AHB I/O space or the AHB Queue Manager, PCI_BAR5 must be used.

Figure 91. PCI-to-AHB Address Translation



10.3.2.8 AHB Master Writes

If the AHB Master Interface receives a Memory Write or Memory Write and Invalidate command from the PCI target, the BAR ID and byte enables are examined to determine the appropriate operation:

- If the BAR ID indicates a CSR access (BAR 4), CSR access is requested and, when granted, the indicated write is performed using the supplied address, data, and byte enables. An AHB master operation is not performed.
- If the BAR ID indicates a memory access (BARs 0-3), the byte enables corresponding to each word of the transfer are examined to determine the AHB address and burst size to use during the transfer. If all byte enables are asserted, an INCR word operation is started on the bus and continues until either the last-data-word indicator is detected, or a word with at least one byte enable de-asserted is encountered. Words received from the Initiator Receive FIFO without all byte enables asserted produce a single byte write operation for each byte that is enabled. PCI Memory writes produce SINGLE or INCR writes on the AHB bus.
- If the BAR ID indicates an I/O access (BAR5), a single word write is performed if all byte enables are asserted, otherwise, a single byte write operation is performed for each byte that is enabled.



All AHB burst write transfers are disconnected at the 8-word address boundary. If the Write FIFO becomes empty during the transfer, the AHB Master Interface disconnects the transfer by driving IDLE cycles on the bus. When data is available again in the FIFO, the bus is re-acquired and the transfer continues.

Data parity errors received from the Receive FIFO are ignored as far as AHB master operations are concerned, however the pci_isr.PPE CSR bit is set to capture the error.

10.3.2.9 AHB Master Reads

If the AHB Master Interface receives a Memory Read, Memory Read Line or Memory Read Multiple command from the PCI target, the BAR ID and byte enables are examined to determine the appropriate operation:

- If the BAR ID indicates a CSR access (BAR 4), CSR access is requested and, when granted, a single word read is performed using the supplied address. An AHB master operation is not performed.
- If the BAR ID indicates a memory access (BARs 0-3), an INCR word read operation is started on the bus and continues until the PCI Initiator Interface indicates that the requesting master has terminated the transfer. Any unused data in the Initiator Read FIFO is discarded.
- If the BAR ID indicates an I/O access (BAR5), a single word read is performed if all byte enables are asserted, otherwise, a single byte read operation is performed for each byte that is enabled.

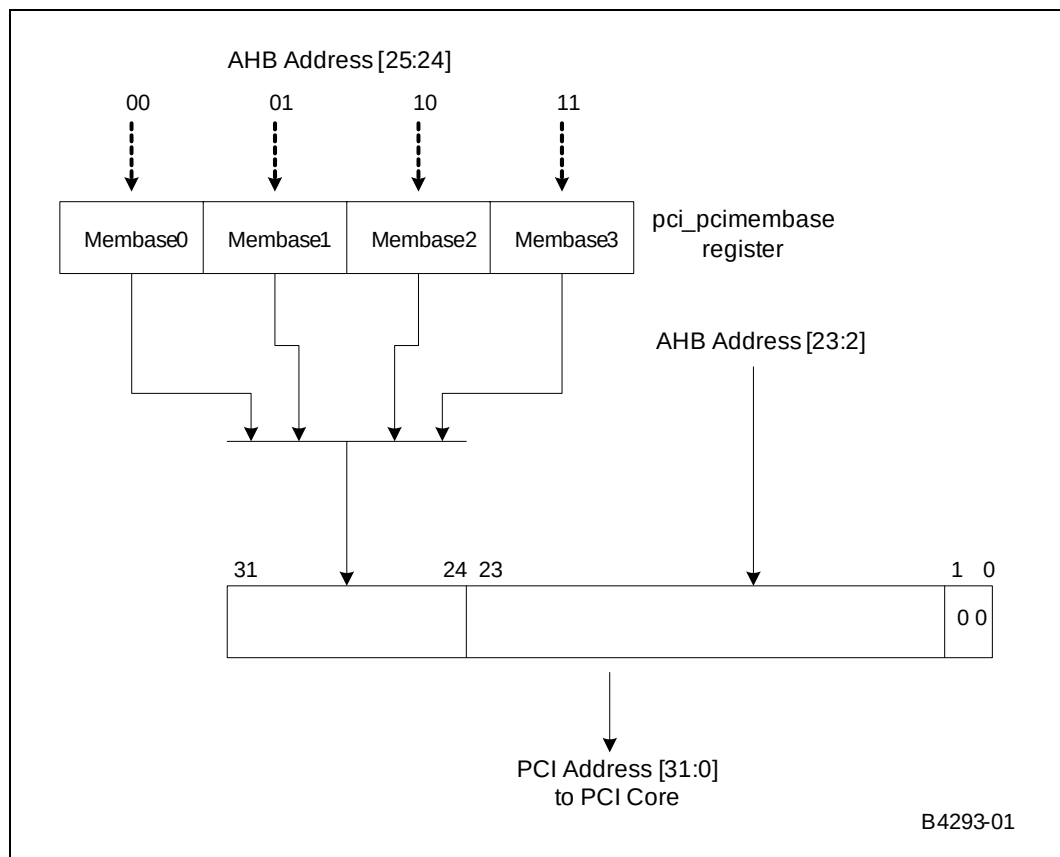
All AHB burst read transfers are disconnected at the 8-word address boundary. If the Read FIFO becomes full during the transfer, the AHB Master Interface disconnects the transfer by driving IDLE cycles on the bus. When room is available in the FIFO again, the bus is re-acquired and the transfer continues.

10.3.2.10 AHB Slave Interface

The AHB Slave Interface provides external AHB masters with read/write access to external PCI targets, PCI Controller PCI Configuration registers, and local PCI Controller CSRs. The interface supports both little-endian and big-endian addressing. AHB Slave accesses are processed by converting the AHB transaction to a PCI transaction with PCI address, Command, and byte enables provided to the PCI Core. The PCI Core then executes the requested PCI transaction and provides status and read data to the AHB Slave.

10.3.2.10.1 AHB-to-PCI Address Translation

For each of the four 16MB memory ranges, the lower 24 bits of the AHB address map to the lower 24 PCI address bits and an 8-bit field in the PCI_PCIMEMBASE register provides the upper 8 address bits of the PCI address. Bits 24 and 25 of the AHB address determine which of the 8-bit fields in PCI_PCIMEMBASE is used. In this manner, the Intel XScale processor can map each range to any 16MB region in the full 4GB PCI address space by appropriately initializing the four base address fields in pci_pcimembase. See [Figure 92](#) for details.

**Figure 92. AHB-to-PCI Address Translation – Memory Cycles****10.3.2.10.2 AHB Slave Read Accesses**

An AHB Slave Read occurs when an external AHB master performs a read operation addressing one of the PCI Controller AHB address regions. The AHB Slave Interface services one PCI read request at a time. When an AHB Slave Read occurs, the Slave Interface checks the status of the Initiator Request FIFO to determine the course of action:

- If the Initiator Request FIFO is not full and a previous direct PCI read is not currently pending, the Slave Interface immediately issues a retry for direct PCI reads and writes the PCI address to the Request FIFO. CSR reads complete immediately with one wait state.
- If the Request FIFO is full or if a previous read request is still pending (has not completed yet on AHB), a retry is issued in response to direct PCI reads and CSR reads. The address of the read is not written to the Request FIFO in the direct PCI read case.
- If the DMA Controller has been granted access to the Request FIFO, a retry is issued in response to direct PCI reads and reads of the four pci_np_xxx CSRs. The address of the read is not written to the Request FIFO in the direct PCI read case. Reads of any of the other CSRs complete immediately with one wait state.

When the PCI read data is available in the Initiator Receive FIFO, the Slave Interface provides the data in response to the retried read request from the same master. While the read is pending, any accesses attempted by other AHB masters will be retried with



no request sent to the PCI Core. The Slave Interface only issues the retry on the cycle following the address phase on AHB, never in the middle of a burst. Bursts of 1 to 255 words are supported.

10.3.2.10.3 AHB Slave Write Accesses

An AHB Slave Write occurs when an external AHB master performs a write operation addressing one of the PCI Controller AHB address regions (indicated by the assertion of `arbs_hsel_pcc` or `arbs_hsel_pcc_mmr` by the AHB arbiter). When an AHB Slave Write occurs, the Slave Interface checks the status of the Initiator Request FIFO and Initiator Transmit FIFOs to determine the course of action:

- If a PCI read is currently pending, a retry is issued on AHB. The request is not written to the Request FIFO.
- If the DMA Controller has been granted access to the Request FIFO, a retry is issued on AHB for direct writes to PCI (`arbs_hsel_pcc_asserted`) or writes to any of the four `pci_np_xxx` registers. Writes to any of the other CSRs complete immediately. The request is not written to the Request FIFO in the direct PCI write case.
- If the operation is a direct write to PCI and either the Request FIFO is full or the Transmit FIFO has insufficient storage available for the entire burst (INCR bursts are assumed to be up to 8 words), a RETRY is issued on AHB and the request is not written to the Request FIFO.
- If the operation is a direct write to PCI and is a SINGLE, INCR with burst length of 8 or less, INCR4 or INCR8, and the Request FIFO is not full and the Transmit FIFO has sufficient storage for all of the data in the entire burst, the request is immediately posted in the Request FIFO, the data is written to the Transmit FIFO and the transfer completes on AHB. In the case of an INCR write of more than eight words, if the Transmit FIFO becomes full during the transfer, wait states will be inserted on the AHB bus until the FIFO becomes not full again.

10.3.2.10.4 AHB Accesses of Local PCI Configuration Registers

The PCI Controller local PCI Configuration registers (listed in [Table 200 on page 550](#)) in the PCI Core are accessed from the AHB bus via a set of configuration port CSRs: `pci_crp_ad_cbe`, `pci_crp_wdata`, and `pci_crp_rdata`.

A read access is processed as follows:

1. An AHB master writes the PCI function number, PCI configuration register word address, and read command to the `PCI_CRP_AD_CBE` register. Note: the PCI Controller has one PCI function so the function number must always be 0.
2. The hardware reads the addressed register in the PCI Core and loads the data into the `PCI_CRP_RDATA` register. During this operation, any access of CSR space from the AHB will be retried.
3. The AHB master reads the `pci_crp_rdata` register to retrieve the data. Note that this AHB read operation can immediately follow the write in step 1. The hardware will retry the read until the data is valid thus making any data validity handshaking transparent to software.

A write access is processed as follows:

1. An AHB master writes the PCI function number, PCI configuration register word address, write command and active-low byte enables to the `PCI_CRP_AD_CBE` register. Note: the PCI Controller has one PCI function so the function number must always be 0.
2. The AHB master writes the data to be written to the `PCI_CRP_WDATA` register.



3. The hardware writes the data in PCI_CRP_WDATA to the enabled bytes in the addressed register of the PCI Core. During this operation, any access of CSR space from the AHB will be retried.

A burst CSR write operation may be used to write PCI_CRP_AD_CBE and PCI_CRP_WDATA to initiate the PCI configuration register write operation.

10.3.2.10.5 AHB Non-Prefetch PCI Accesses

The PCI Controller can generate “non-prefetch” (single) PCI cycles such as I/O Read and Write and Configuration Read and Write cycles from the AHB bus via the non-prefetch CSRs PCI_NP_AD, PCI_NP_CBE, PCI_NP_WDATA, and PCI_NP_RDATA.

A read access is processed as follows:

1. An AHB master writes the 32-bit address of the PCI read cycle to the PCI_NP_AD register.
2. The AHB master writes the PCI Command Type and data byte enables for the desired read cycle to the PCI_NP_CBE register.
3. The hardware sends the read request (address, command, byte enables) to the PCI Core which performs the indicated read cycle and returns the read data. The data is loaded into the pci_np_rdata register. While the read operation is pending, any access of CSR space from the AHB will be retried.
4. The AHB master reads the PCI_NP_RDATA register to retrieve the data. Note that this AHB read operation can immediately follow the write in step 2. The hardware will retry the read until the data is valid thus making any data validity handshaking transparent to software.

A write access is processed as follows:

1. An AHB master writes the 32-bit address of the PCI write cycle to the PCI_NP_AD register.
2. The AHB master writes the PCI Command Type and data byte enables for the desired write cycle to the PCI_NP_CBE register. Byte enables conform to the PCI little-endian convention.
3. The AHB master writes the data to be written to the PCI_NP_WDATA register.
4. The hardware sends the write request (address, command, byte enables, data) to the PCI Core which performs the indicated write cycle. While the write operation is in progress, any access of CSR space from the AHB will be retried.

It should be noted that the AHB Slave Interface and CSR hardware do not interpret the contents of the non-prefetch registers. The address, command, byte enables, and write data are passed to the PCI Core as-is. Thus, for example, I/O read and write requests must be set-up such that the byte enables are consistent with the 2 LSBs of the address in accordance with the PCI Local Bus Specification.

10.3.2.11 PCI Byte Enable Generation

The byte enables for single PCI transactions are generated based on the type of AHB access (direct read/write or non-prefetch read/write), address, transfer size (8-bit, 16-bit, 32-bit), and the settings in effect for AHB endianness and data swapping modes. All 32-bit accesses obviously assert all byte enables when the transaction is sent to the PCI Initiator. Since the AHB Slave Interface only supports bursts with a size of 32-bits, PCI burst transactions will always have all byte enables asserted. [Table 198](#) shows the byte enables for 8-bit and 16-bit single PCI read and write cycles generated by direct access of the AHB Slave Interface. PCI cycles generated by the non-prefetch CSRs use the value of the NP_BE field of the PCI_NP_CBE CSR for the byte enables.

**Table 198. PCI Byte Enables for Sub-word Single AHB Read/write Cycles**

AHB addr[1:0]	pci_csr.ABE	pci_csr.ADS	PCI Byte Enable [3:0]	
			8-bit transfer	16-bit transfer
00	0	0	1110	1100
01	0	0	1101	1100
10	0	0	1011	0011
11	0	0	0111	0011
00	0	1	0111	0011
01	0	1	1011	0011
10	0	1	1101	1100
11	0	1	1110	1100
00	1	0	0111	0011
01	1	0	1011	0011
10	1	0	1101	1100
11	1	0	1110	1100
00	1	1	1110	1100
01	1	1	1101	1100
10	1	1	1011	0011
11	1	1	0111	0011

10.3.3 PCI Controller DMA

The IXP45X/IXP46X network processors contain two channels that can be used for DMA (Direct Memory Accesses) to/from the PCI bus and the AHB. The DMA Controller function provides two channels of DMA capability to off load, from the Intel XScale processor, large data transfers between the PCI bus and AHB.

The DMA channels are unidirectional: one DMA channel is used for PCI-to-AHB transfers and one DMA channel is used for AHB-to-PCI transfers. The DMA transfers are implemented using three of the PCI Controller Configuration and Status Registers to specify the PCI address, the AHB address, and the transfer length/control. Each DMA channel has two sets of three registers to provide buffering for consecutive transfers.

For each direction, when a DMA channel is executing one transfer using the active DMA register set, the other DMA register set can be set-up by the Intel XScale processor to specify the next transfer. Both DMA channels can run concurrently so that individual PCI-to-AHB transfers and AHB-to-PCI transfers that make up the DMA transfers are interleaved on the AHB and PCI bus.

Individual DMA-complete and DMA-error status indication is provided for each channel using the DMA Control Register (PCI_DMACTRL) with an interrupt that may be optionally generated in each case.

The DMA channels share resources with the AHB Master and Slave interfaces and therefore must arbitrate for these resources. AHB-to-PCI DMA transfers use the AHB Master Interface, the PCI Initiator Request, and Initiator Transmit FIFOs. PCI-to-AHB DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Receive FIFO. Use of the AHB Master Interface will revolve between the two DMA channels and PCI requests that appear in the Target Receive FIFO.



While a particular channel is accessing the Initiator Request FIFO, accesses to the PCI bus coming in to the AHB Slave Interface from the AHB will be retried. The access will be flagged by the hardware to signal the DMA Controller channels that a PCI access is pending (AHB masters must attempt retried transfers until complete). This enables the DMA channels to permit the AHB initiated PCI access to go through to the PCI bus.

Additionally, while the AHB Master Interface is in use by a DMA channel, PCI requests that appear in the Target Receive FIFO are flagged to allow these received requests to gain access of the AHB bus.

Access to CSRs from the AHB bus is unrestricted while the DMA channels are operating. Ability to access the PCI Controller Control and Status Registers is provided to allow the Intel XScale processor to set up the off-line DMA Register set while the on-line DMA Register set is operating.

The register sets associated with the DMA channels are as follows:

1. PCI-to-AHB Transfers
 - a. Register Set 0
 - PCI-to-AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR)
 - PCI-to-AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR)
 - PCI-to-AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH)
 - b. Register Set 1
 - PCI-to-AHB DMA AHB Address Register 1 (PCI_PTADMA1_AHBADDR)
 - PCI-to-AHB DMA PCI Address Register 1 (PCI_PTADMA1_PCIADDR)
 - PCI-to-AHB DMA Length Register 1 (PCI_PTADMA1_LENGTH)
2. AHB-to-PCI Transfers
 - a. Register Set 0
 - AHB-to-PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR)
 - AHB-to-PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR)
 - AHB-to-PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH)
 - b. Register Set 1
 - AHB-to-PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR)
 - AHB-to-PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR)
 - AHB-to-PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH)

The PCI Address Registers described above are used to specify the beginning 32-bit word address for the PCI side of the DMA transfers. The AHB Address Registers, described above, are used to specify the beginning 32-bit word address for the AHB side of the DMA transfers. The least significant two bits of both addresses are hard-wired to logic 0. Thus, all transfers are word-aligned.

The Length Registers are used for three purposes:

- Sixteen bits to define a word count
Bits 15:0 of the Length Registers define the word count.
- One bit to enable the DMA transfer
Bit 31 of the Length Registers enables the DMA transfer to execute. When bit 31 of the Length Registers is set to logic 1, the DMA transfer executes until a word count of zero is reached. When the word count reaches zero and bit 31 of the Length Registers is set to logic 1, bit 31 of the Length Register is cleared to logic 0. When bit 31 of the Length Register is set to logic 0, the register set associated with the

DMA channel is disabled. The second register set may be active and using the DMA channel when the first DMA has finished.

- One bit to define the byte order of the data transferred.
Bit 28 of the Length Register is used to provide a byte swap on the DMA data as data is transferred from the AHB to the PCI bus or from the PCI Bus to AHB, depending upon the direction of the DMA transfer. When bit 28 is set to logic 1, a byte swap will occur on the DMA data. Figure 93 and Figure 94 demonstrates the DMA transfer byte lane swapping.

Figure 93. AHB-to-PCI DMA-Transfer Byte Lane Swapping

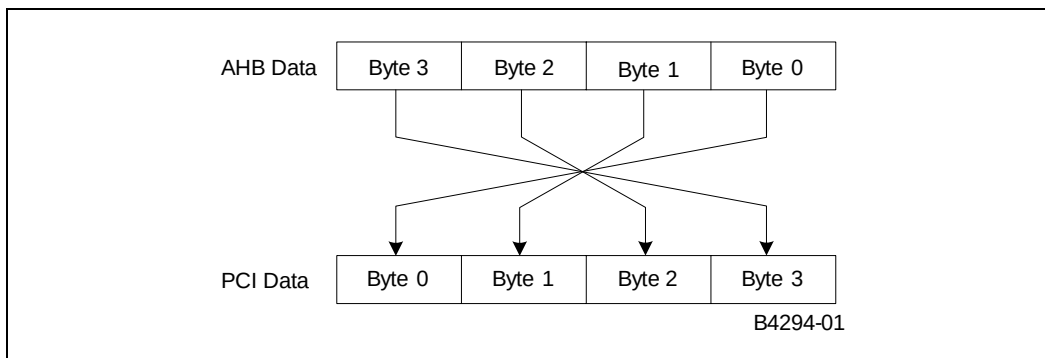
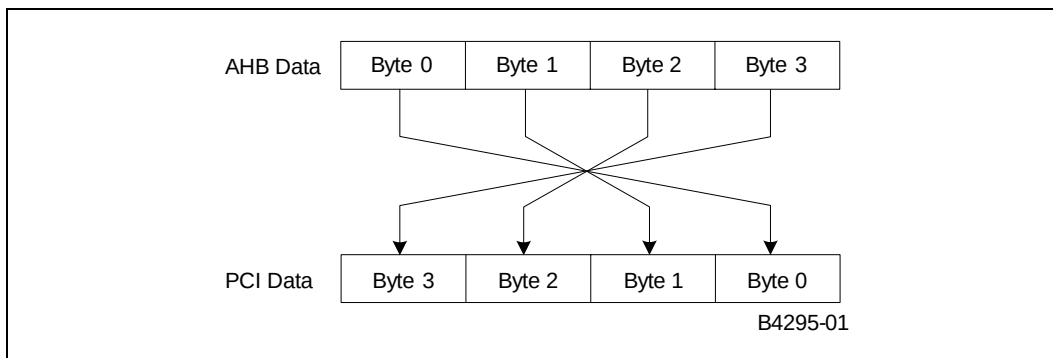


Figure 94. PCI-to-AHB DMA-Transfer Byte Lane Swapping



The DMA channels share resources with the AHB Master and Target interfaces and therefore must arbitrate for these resources. AHB-to-PCI DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Transmit FIFO. PCI-to-AHB DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Receive FIFO. Use of the AHB Master Interface will revolve between the two DMA channels and PCI requests that appear in the Target Receive FIFO.

A DMA transfer is started on a particular channel by writing the PCI start address, AHB start address, and length to one set of DMA CSRs. If the channel enable bit is set in the length register, that DMA channel is enabled and the transfer is executed. While the transfer is taking place, the other set of DMA CSRs can be set-up to specify the next transfer. When the current transfer is complete, the DMA complete bit is set in the status register and the channel enable bit is cleared in the length register. Execution then starts using the second set of DMA CSRs, if the channel is enabled in the length register. Transfers continue in this fashion until the channel enable bits in both sets of DMA length registers are cleared.



The DMA channels use 8-word burst accesses on the PCI and AHB busses (PCI-to-AHB and AHB -to-PCI) whenever possible. In the general case, a transfer will issue a starting burst from 1 to 7 words to align the AHB word address to an 8-word boundary. Then issue 8-word bursts until the words remaining to be transferred is less than 8. Then complete the transfer with a burst of from 1 to 7 words. Eight-word bursts are the maximum sustained length that the AHB — and the PCI bus — can transfer. Every eight words, the PCI bus will disconnect and reconnect later.

This implementation allows fairness among all devices on the PCI bus. In the general case, a transfer will issue a beginning burst transfer from one to eight words to align the AHB word address of the DMA to an eight-word boundary.

The subsequent transfers will be issued as eight-word bursts until the words remaining to be transferred are eight words or less. The final transfer will complete the DMA with a burst of one to eight words.

The example below demonstrates how to use the DMA channels.

The goal is to:

- Write a 16-word burst to the PCI Bus with no byte-swapping, using the AHB-to-PCI DMA channel
 - Initialize a 16-word burst read from the PCI Bus, using the PCI-to-AHB DMA channel
 - Initialize a six-word burst write to the PCI Bus using the AHB-to-PCI DMA channel. The AHB-to-PCI DMA channel is used to complete PCI Memory Cycle write accesses and the PCI-to-AHB DMA channel is used to complete PCI Memory Cycle read accesses always.
1. Update the AHB-to-PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR) with PCI_ATPDMA0_AHBADDR = 0x00004000 and the AHB-to-PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR) with PCI_ATPDMA0_PCIADDR = 0xFC000004.
 2. Update the AHB-to-PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR) with PCI_ATPDMA1_AHBADDR = 0x00004F00 and the AHB-to-PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR) with PCI_ATPDMA1_PCIADDR = 0xA2000004.
 3. Update the PCI-to-AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR) with PCI_PTADMA0_AHBADDR = 0x00004A00 and the PCI-to-AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR) with PCI_PTADMA0_PCIADDR = 0x10000004.
 4. Update the AHB-to-PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH) with PCI_ATPDMA0_LENGTH = 0x80000010.
The DMA write transfer to the PCI bus begins.
 5. Update the PCI-to-AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH) with PCI_PTADMA0_LENGTH = 0x80000010.
Assume that this DMA channel is enabled prior to the end of the first eight-word burst of the first write DMA transfer ending. The DMA read transfer to the PCI bus becomes interleaved with the first write transfer. So the first eight words of the read starts towards completion.
 6. Update the AHB-to-PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH) with PCI_ATPDMA1_LENGTH = 0x90000006.
Assume this is set while the above read DMA transaction is occurring.
 7. The next PCI transfer is completing the last eight words of the initial 16-word write DMA transfer. That is followed by the last eight words of the 16-word read DMA transfer and the execution of the six-word write transfer with the data byte lanes swapped.



10.3.3.1 AHB-to-PCI DMA Channel Operation

The AHB-to-PCI (ATP) channel uses the PCI Core Initiator Request and Initiator Transmit FIFOs. The channel reads data from the AHB bus and writes it to a PCI target on word aligned boundaries.

A DMA transfer from AHB-to-PCI is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_ATPDMA0/1_PCIADDR, PCI_ATPDMA0/1_AHBADDR, PCI_ATPDMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_ATPDMA0/1_LENGTH register, the DMA transfer commences.
2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for the Transmit FIFO to become empty.
3. The DMA Controller requests access to the AHB Master Interface which it shares with the other DMA channel and accesses from the PCI bus.
4. When access is obtained, data is read from AHB and loaded into the Initiator Transmit FIFO. A PCI write request is loaded into the Initiator Request FIFO.
5. The AHB Master and Slave Interfaces are released.
6. When the transfer completes on the PCI bus, the DMA address and length registers are updated.
7. Steps 2-6 are repeated until all words are transferred. When done, the channel enable bit in the PCI_ATPDMA0/1_LENGTH register is cleared, the DMA complete status bit is set.
8. In response to the interrupt, an AHB agent may read the DMA Control register (pci_dmactrl) to determine the status of the transfer.

10.3.3.2 PCI-to-AHB DMA Channel Operation

The PCI-to-AHB (PTA) channel uses the PCI Core Initiator Request and Initiator Receive FIFOs. The channel reads data from the PCI bus and writes it to an AHB slave on word aligned boundaries. A DMA transfer from PCI-to-AHB is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_PTADMA0/1_PCIADDR, PCI_PTADMA0/1_AHBADDR, PCI_PTADMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_PTADMA0/1_LENGTH register, the DMA transfer commences.
2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for non-full indication from the Initiator Request FIFO.
3. The DMA Controller issues a read request to the PCI Core via the Request FIFO and releases the AHB Slave Interface.
4. When the PCI data arrives in the Initiator Receive FIFO, the DMA Controller requests access to the AHB Master Interface which it shares with the AHB-to-PCI DMA channel and accesses from the PCI bus.
5. When access is obtained, data is read from the Target Receive FIFO and written to the AHB agent.
6. When the transfer completes on the AHB bus, the DMA address and length registers are updated.
7. Steps 2-6 are repeated until all words are transferred. When done, the channel enable bit in the PCI_PTADMA0/1_LENGTH register is cleared, the DMA complete status bit is set.
8. In response to the interrupt, an AHB agent may read the DMA Control register (PCI_DMACTRL) to determine the status of the transfer.



10.3.4 Data Byte Alignment and Addressing — PCI Endianness

The *PCI Local Bus Specification*, Rev. 2.2 defines the byte-addressing convention on the PCI Bus as little-endian. Since the byte addressing convention on the PCI bus is little-endian and the convention used on the AHB bus is big-endian, data passing from the PCI Core FIFOs to and from the AHB bus can go through an optional CSR-bit controlled byte lane reversal. The AHB bus interfaces can also be configured to operate in either big-endian or little-endian addressing conventions.

As shown in [Figure 95](#), when an external PCI device accesses an AHB address with the AHB in big-endian mode (`pci_csr.ABE = 1`), the `pci_csr.PDS` (PCI Data Swap) bit controls the byte lane swapping between the two busses. When `pci_csr.PDS` is 1, the PCI data bytes are treated as little-endian addressed and are swapped to the corresponding big-endian byte lanes of the AHB bus. In the figure, PCI bytes are numbered according to the corresponding PCI byte enables and AHB bytes are numbered according to the corresponding byte address on the AHB bus. For example, a PCI write with byte enable 2 asserted and `pci_csr.PDS = 1` will produce an AHB byte write with AHB address bits `[1:0] = 10b`. When `pci_csr.PDS` is 0, the PCI data bytes are treated as big-endian addresses and are not swapped. In this case, a PCI write with PCI byte enable 2 asserted will produce an AHB byte write with AHB address bits `[1:0] = 01b`.

[Figure 96](#) shows the byte alignment in AHB little-endian mode (`pci_csr.ABE = 0`).



Figure 95. Byte Lane Routing During PCI Target Accesses of the AHB Bus – Big-Endian AHB Bus

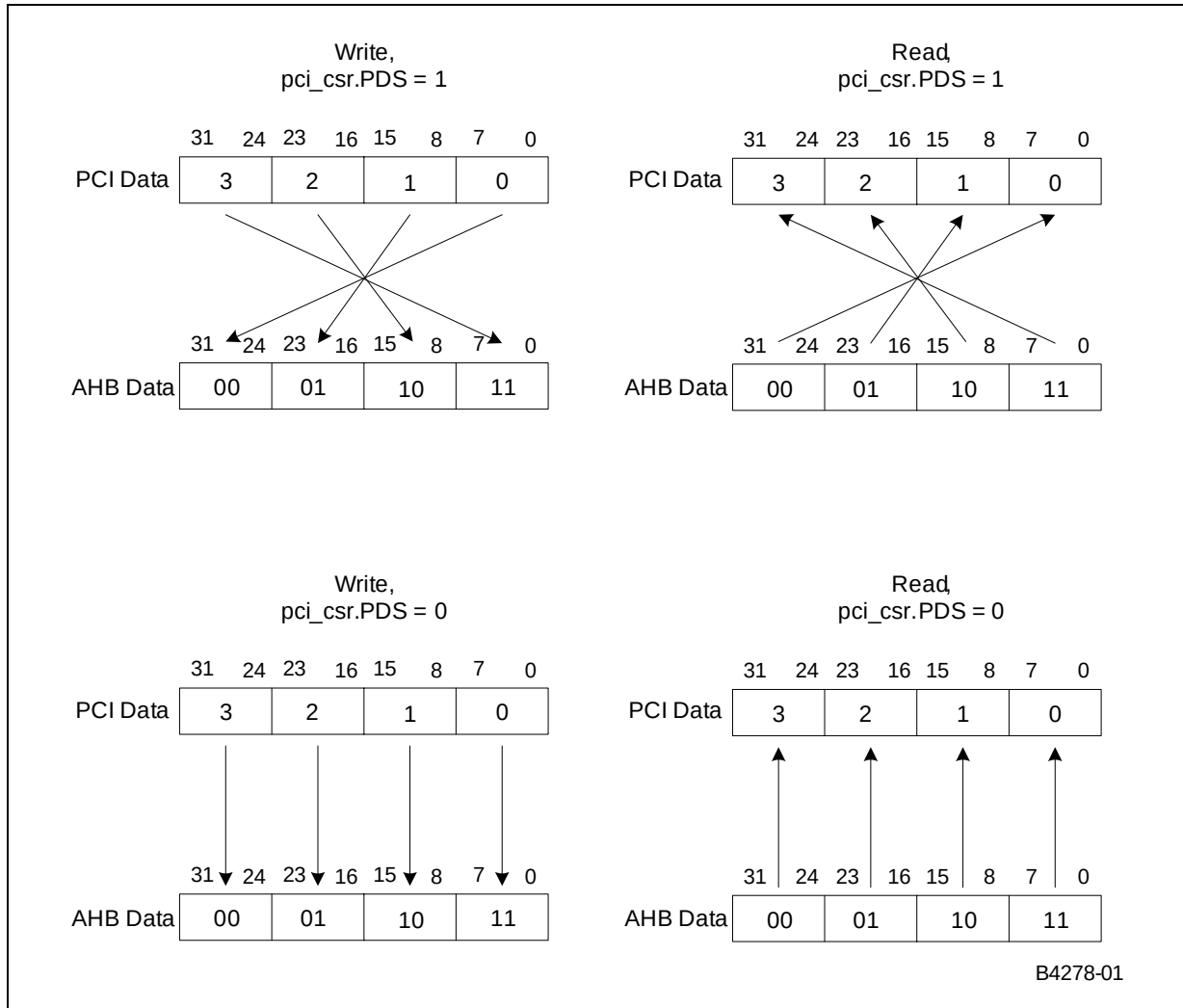
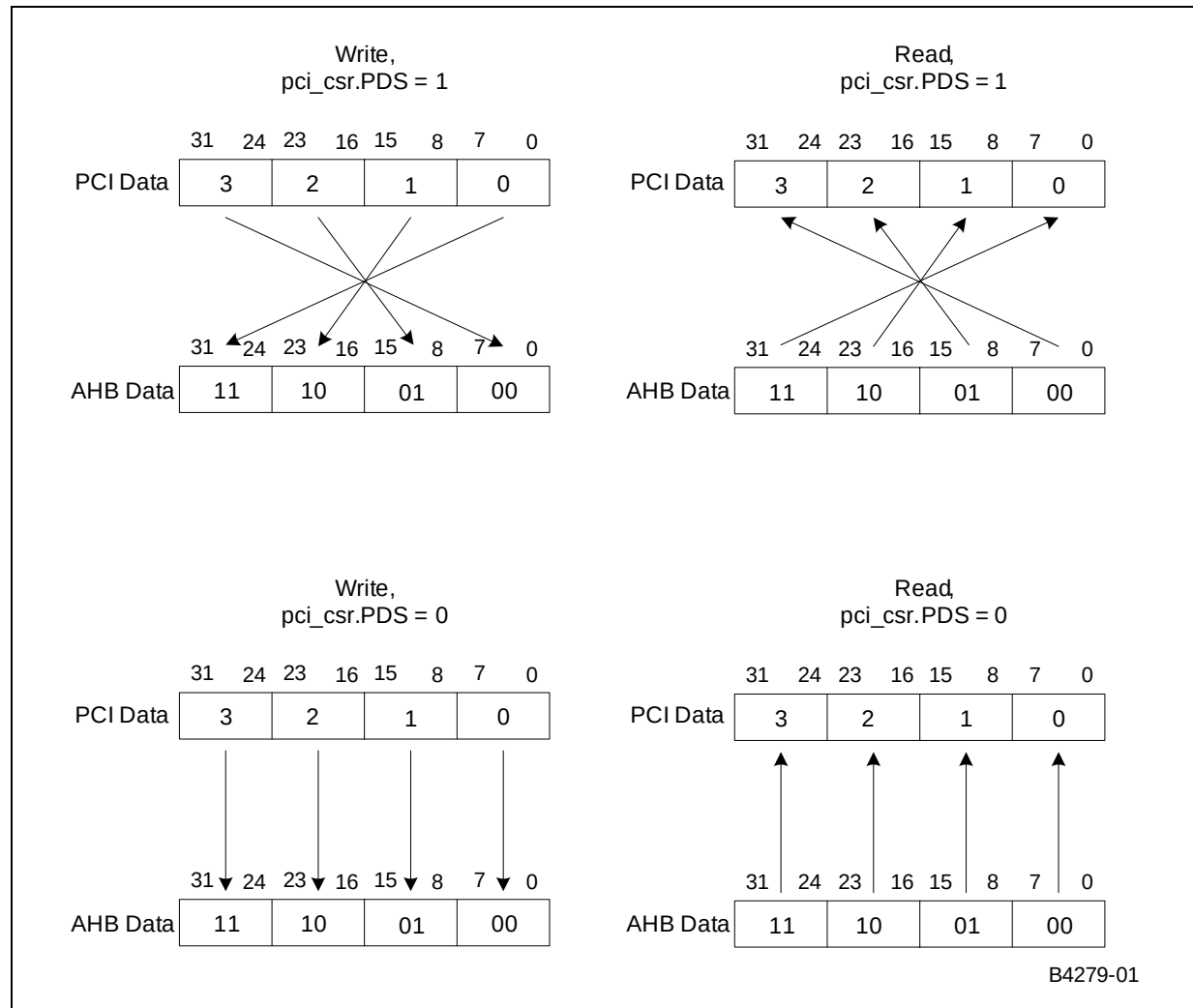




Figure 96. Byte Lane Routing During PCI Target Accesses of the AHB Bus – Little-Endian AHB Bus



In a similar manner, the `pci_csr.ADS` bit controls byte lane routing when an AHB agent accesses an external PCI device using memory-mapped accesses (no swapping is performed during the CSR-controlled non-prefetch cycles). [Figure 97](#) and [Figure 98](#) illustrate the data routing in AHB big-endian and little-endian modes respectively.



Figure 97. Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Big-Endian AHB Bus

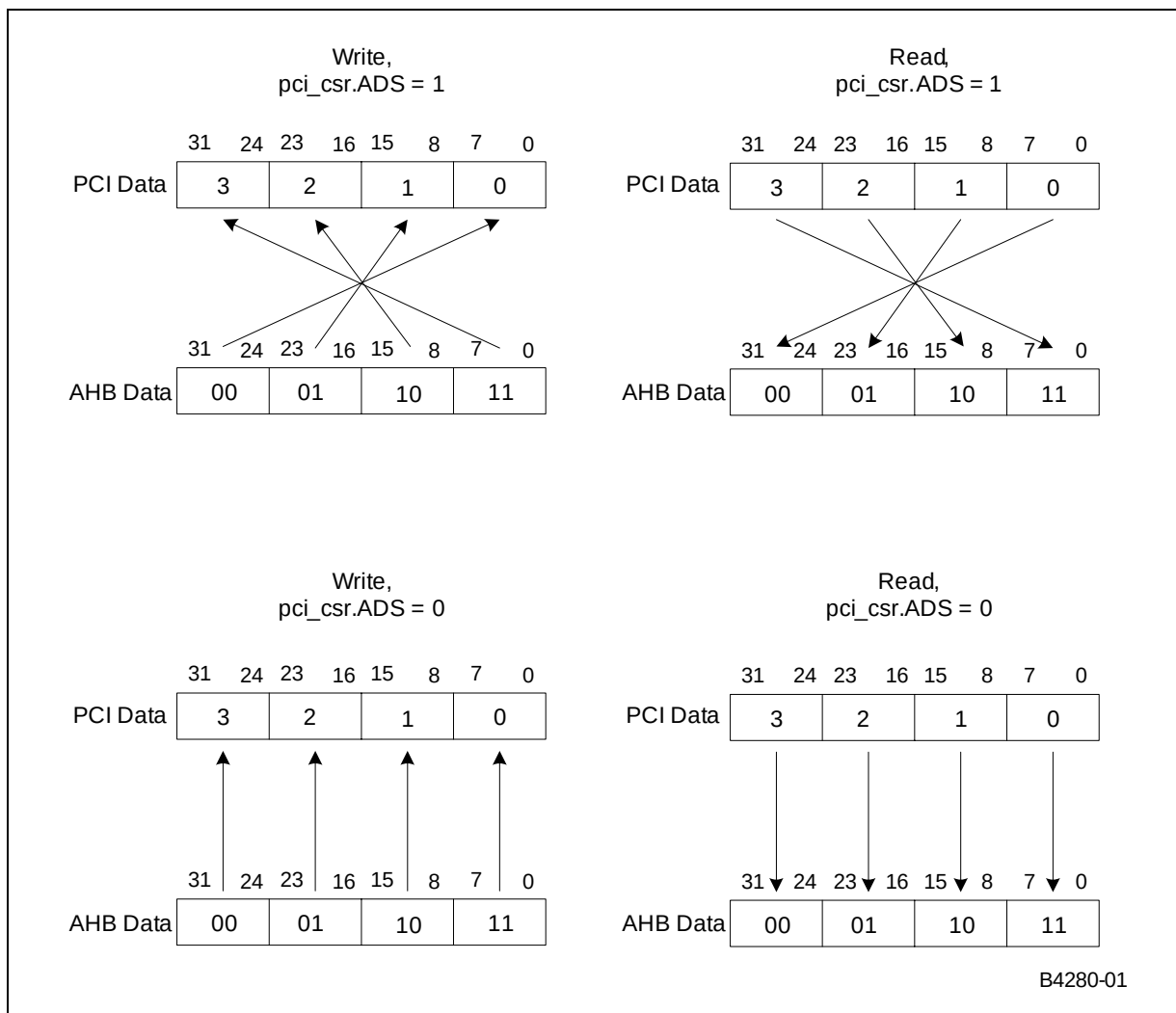
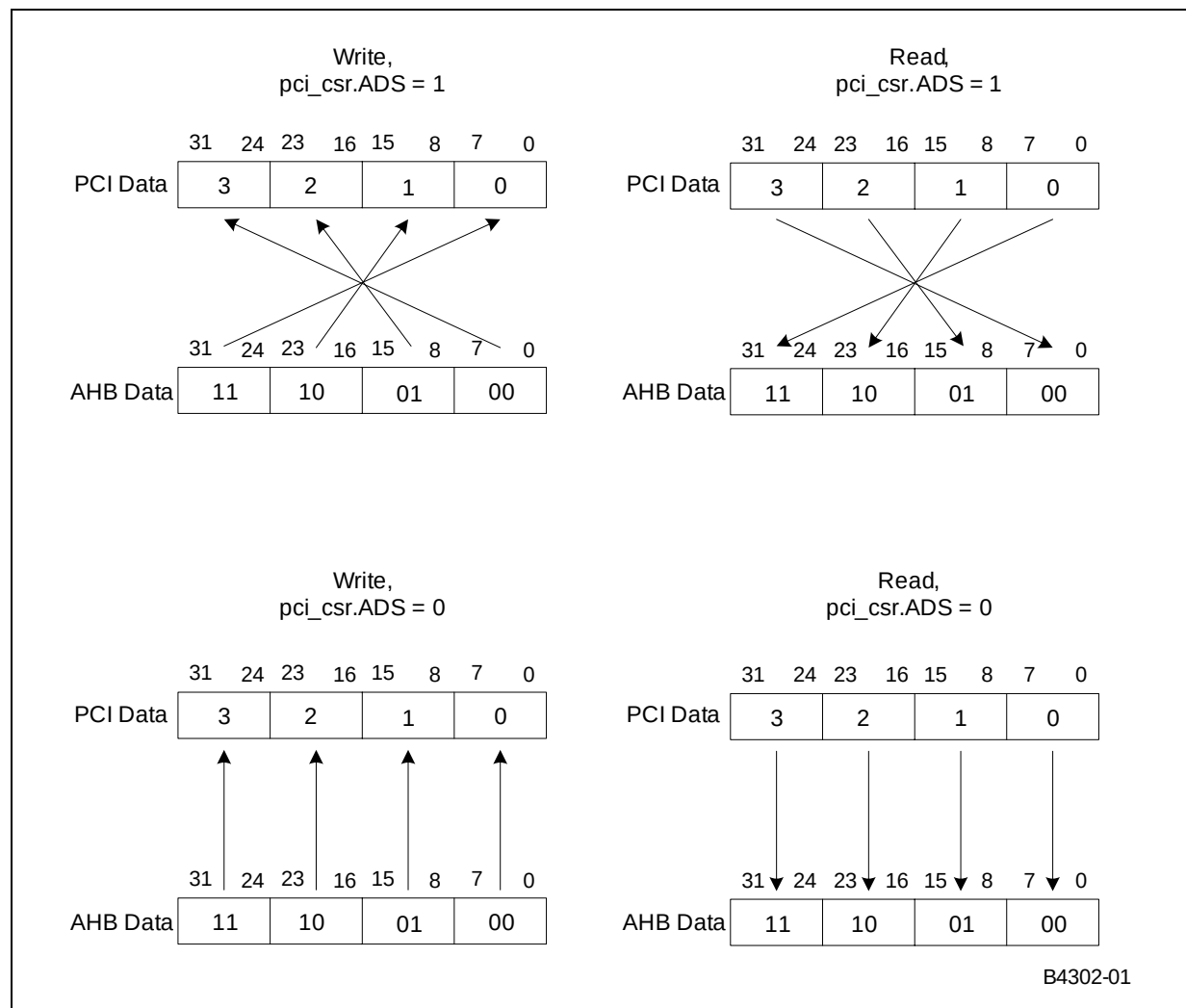




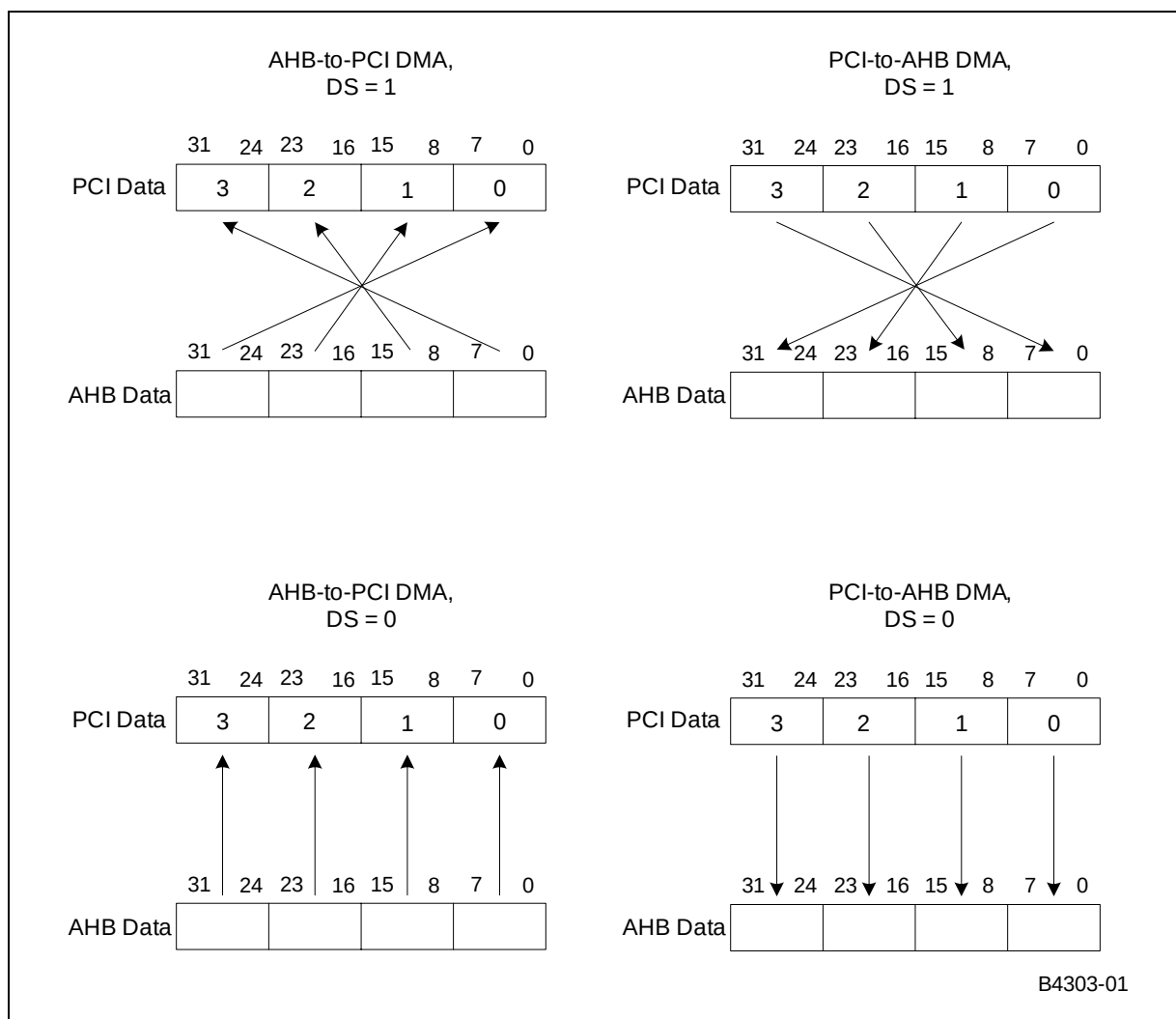
Figure 98. Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Little-Endian AHB Bus



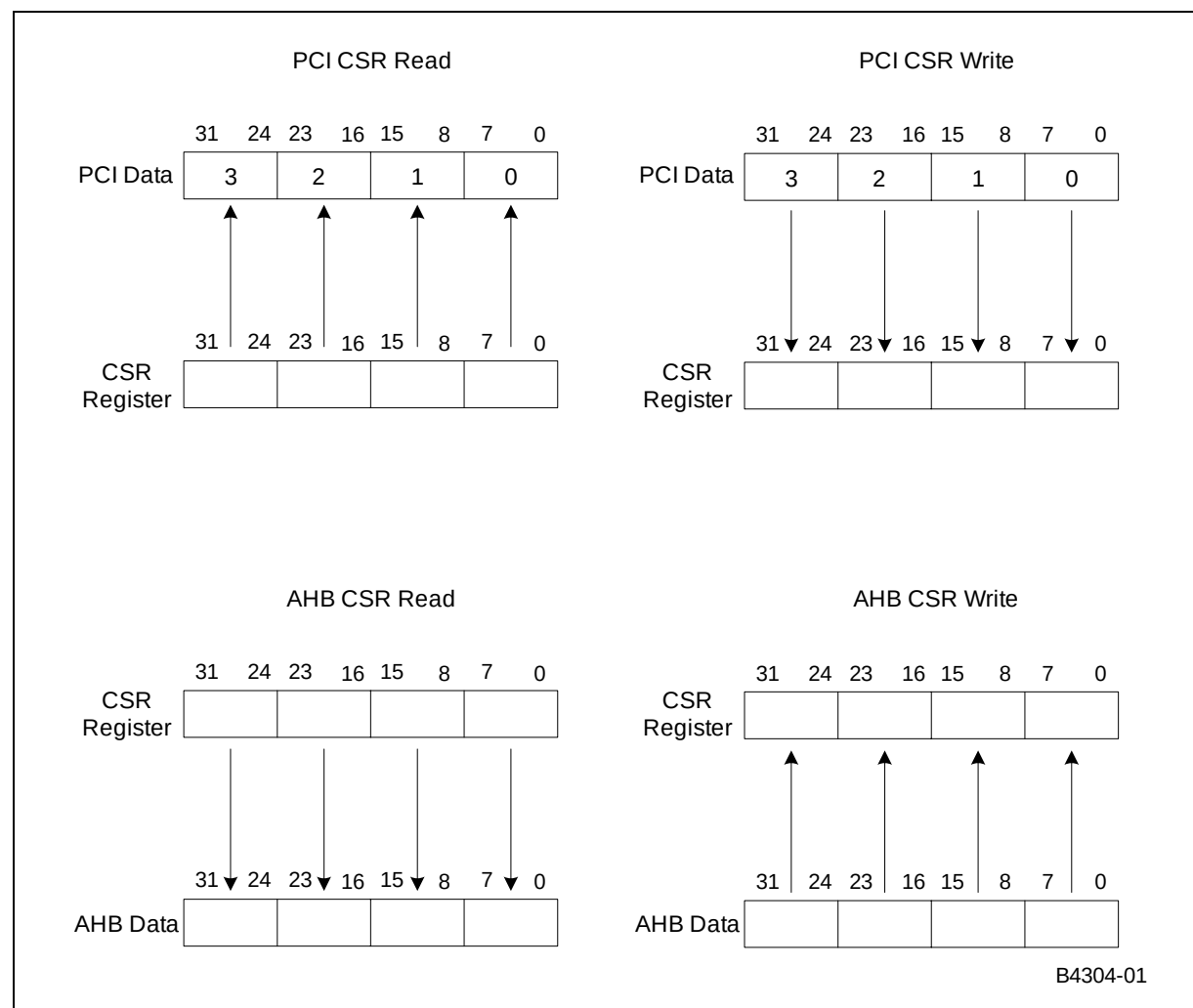
During DMA transfers, byte lane routing is controlled by the DS bit in the DMA length register as shown in Figure 99.



Figure 99. Byte Lane Routing During DMA Transfers



There is no byte lane reversal for accesses to PCI Controller CSRs or PCI Configuration registers. [Figure 100](#) shows the byte lane routing for these types of accesses. A PCI CSR write with byte enable 2 asserted, for example, will write to bits 23:16 of the register. An AHB write with Address bits [1:0] = 10b in big-endian mode (`pci_csr.ABE = 1`) will write bits 15:8 of the register. In little-endian mode (`pci_csr.ABE = 0`), bits 23:16 will be written.

**Figure 100. Byte Lane Routing During CSR Accesses**

10.3.5 PCI Controller Interrupts

The PCI Controller supports generation of PCI interrupts and interrupts to internal AHB agents. Complete control of the interrupt sources and enabling is provided using two registers: the PCI Interrupt Status Register (PCI_ISR) and PCI Interrupt Enable Register (PCI_INTEN).

10.3.5.1 PCI Interrupt Generation

The PCI Door Bell Register (PCI_PCIDORBELL) is used to generate a PCI interrupt on the PCI Bus using the PCI_INTA_N signal. This register is read/write-1-to-set from the AHB bus, and read/write-1-to-clear from the PCI bus. All bits are ORed together to generate the PCI interrupt. The sequence is:

- An AHB agent writes a pattern of ones to the PCI_PCIDORBELL register, setting the corresponding bits in the register.
- The interrupted PCI device reads the bit pattern in the doorbell register and writes the same pattern back to clear the bits and deassert the interrupt.



The PCI interrupt is enabled by the PDB bit of the Interrupt Enable Register (`pci_inten`). When this bit is set and at least 1 bit is set in the `pci_pcidoorbell` register, an interrupt is asserted on the `PCI_INTA_N` open-drain output.

The PDB bit — Bit 7 of the PCI Interrupt Enable Register (`PCI_INTEN`) — is used to enable the external PCI Interrupt. When bit 7 is set to logic 1, the external PCI Interrupt logic is enabled, an interrupt is asserted on the `PCI_INTA_N` open-drain output. When bit 7 is set to logic 0 the external PCI Interrupt logic is disabled.

Bit 7 of the PCI Interrupt Status Register (`PCI_ISR`) displays the status of the external PCI Interrupt. This bit will be set to logic 1 when any of the `PCI_PCIDOOORBELL` bits are set to logic 1.

10.3.5.2 Internal Interrupt Generation

The PCI Controller employs three internal interrupt outputs to signal AHB agents of the occurrence of various events. The `PCC_INT` signal is a general-purpose interrupt, while `PCC_ATPDMA_INT`, and `PCC_PTADMA_INT` are DMA interrupts. All interrupts are high-active and remain asserted until an AHB agent clears the interrupting source by writing appropriate CSR register bits in the PCI Controller.

The general-purpose interrupt `pcc_int` is can be asserted when:

- A PCI error occurs
- An AHB error occurs
- An AHB-to-PCI DMA transfer is complete or terminates due to an error
- A PCI-to-AHB DMA transfer is complete or terminates due to an error
- A doorbell is “pushed” by an external PCI device

The `PCI_ISR` register indicates the source(s) of the `PCC_INT` interrupt. The `PCC_INTEN` register provides an enable for each of the sources in `PCI_ISR`. If a bit is set in `PCI_ISR` and its corresponding enable is set in `PCI_INTEN`, the `PCC_INT` output will be asserted high-active. The interrupt remains asserted until either the source in `PCI_ISR` or the enable in `PCI_INTEN` is cleared. Clearing an interrupt source may involve clearing bits in other CSR registers.

The `PCI_AHBDORBELL` register is used to generate the doorbell interrupt to an AHB agent. This register is read/write-1-to-set from the PCI bus, and read/write-1-to-clear from the AHB bus. All bits are ORed together to generate the interrupt. The sequence is:

- an external PCI agent writes a pattern of ones to the `PCI_AHBDORBELL` register, setting the corresponding bits in the register and asserting the interrupt to the AHB agent.
- the AHB agent reads the bit pattern in the doorbell register and writes the same pattern back to clear the bits and deassert the interrupt.

The DMA interrupt `PCC_ATPDMA_INT` is asserted when the `APDCEN` of the DMA Control register is set and the DMA transfer completes or terminates due to an error. DMA completion, with or without error, is indicated by either `APDC0` or `APDC1` bits being set, where 0/1 corresponds to the DMA register set used for the transfer (`PCI_ATPDMA0/1_xxx`).

Likewise, the DMA interrupt `PCC_PTADMA_INT` is asserted when the `PADCEN` of the DMA Control register is set and the DMA transfer completes or terminates due to an error. DMA completion, with or without error, is indicated by either `PADC0` or `PADC1` bits being set, where 0/1 corresponds to the DMA register set used for the transfer (`PCI_PTADMA0/1_xxx`).



Note that DMA completions can generate an interrupt on the DMA interrupt outputs or on the general-purpose interrupt PCC_INT. Separate DMA interrupt enables are provided for each interrupt output signal but there remains just one interrupt source for each DMA channel in the DMA Control register.

10.4 PCI RCOMP Circuitry

The PCI RCOMP circuitry dynamically compensates for variations in operating conditions due to process, temperature and voltage. These variations are measured through a resistive mechanism in a special I/O Pad and evaluated in the associated compensation circuitry. Adjustments are made to the drive strength of the buffers for the PCI interface ensuring error free operation over the entire range of operating conditions.

The RCOMP circuitry requires an external reference resistor that models the load the PCI pads will see in the board environment. For specific RCOMP pin requirements, see the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet*. The circuitry adjusts the PCI pads' current sourcing strength by comparing the voltage of the output buffer driven through the external reference resistor with an internally generated 60% threshold voltage. The circuitry adjusts the PCI pads' current sinking strength by comparing the output buffer voltage with an internally generated 40% threshold voltage. Once drive strengths are determined for the 60% and 40% thresholds a multiplier is applied to the drive strengths to provide for a margin above and below the 60% and 40% thresholds, respectively.

10.5 Register Descriptions

The following sections describe the internal PCI Configuration registers. Registers or fields defined as reserved return 0s when read and are not affected by writes. The access key is as follows:

Table 199. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

10.5.1 PCI Configuration Registers

These registers comprise the configuration registers as defined in the PCI 2.2 specification with the exception of the pci_rtotto register which is device specific. They are accessible from the PCI bus using configuration read and write transactions and from the Intel XScale processor by accessing the PCI Controller CSR-based PCI Configuration register port. [Table 200](#) lists the registers.



Table 200. PCI Configuration Register Map

Offset Address	Register Name	Description	Reset Value	Page
0x00	pci_didvid	Device ID/Vendor ID	0x85018086	550
0x04	pci_srcr	Status Register/Control Register	0x02a00000	551
0x08	pci_ccrid	Class Code/Revision ID	0x0b4000XY	552
0x0c	pci_bhlc	BIST/Header Type/Latency Timer/Cache Line	0x00000000	553
0x10	pci_bar0	Base Address 0	0x00000008	553
0x14	pci_bar1	Base Address 1	0x00000008	554
0x18	pci_bar2	Base Address 2	0x00000008	554
0x1c	pci_bar3	Base Address 3	0x00000008	555
0x20	pci_bar4	Base Address 4	0x00000008	555
0x24	pci_bar5	Base Address 5	0x00000001	556
0x28	(Reserved)			
0x2c	pci_sidsvid	Subsystem ID/Subsystem Vendor ID	0x00000000	556
0x30-38	(Reserved)			
0x3c	pci_latent	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line	0x00000100	557
0x40	pci_rtotto	Defines retry timeout and trdy timeout parameters	0x00008080	557

10.5.2 PCI Configuration Register Descriptions

10.5.2.1 Device ID/Vendor ID Register

Register Name:						pci_didvid																													
Block Base Address:						N / A						Offset Address						0x00						Reset Value						0x85018086					
Register Description:						Provides Device ID and Vendor ID values as specified in the PCI 2.2 Local Bus Specification.																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
DeviceID																VendorID																			

Register		pci_didvid			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	DeviceID	Unique device identifier assigned by Intel. The state of the tlu_pci_id_sel input determines which device Id is used.	0x8501	RO	RO
15:0	VendorID	Unique vendor identifier assigned to Intel by PCISIG	0x8086	RO	RO



10.5.2.2 Status Register/Control Register

Register Name:						pci_srcr																													
Block Base Address:						0xC00000						Offset Address						0x04						Reset Value						0x02a00000					
Register Description:						Contains the Command and Status registers as specified in the PCI 2.2 Local Bus Specification																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
DPE	SSE	RMA	RTA	STA	DEVSEL	MDPE	FBBC	UDF	66MHZ	CLI	(Reserved)											FBBE	SER	SC	PER	PSE	MWIE	SCE	BME	MAE	IOAE				

Register		pci_srcr (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	DPE	Detected Parity Error. Set when this device detects a parity error on the bus even when parity handling is disabled. Writing a 1 to this bit clears it.	0	RW1C	RW1C
30	SSE	Signaled System Error. Set when this device generates a System Error PCI_SERR_N. Writing a 1 to this bit clears it.	0	RW1C	RW1C
29	RMA	Received Master Abort. Set by this device as a Master when its transaction terminates due to a master abort (except for special cycles). Writing a 1 to this bit clears it.	0	RW1C	RW1C
28	RTA	Received Target Abort. Set by this device as a Master when its transaction is terminated due to a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C
27	STA	Signaled Target Abort. Set by this device as a Target when it terminates a transaction with a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C
26:25	DEVSEL	Defines the DEVSEL speed for this device. Set to medium.	01	RO	RO
24	MDPE	Master Data Parity Error. Set by this device as a Master if PER (bit 6) is set and this device either asserted the PCI_PERR_N signal or saw PCI_PERR_N asserted for one of its data phases.	0	RW1C	RW1C
23	FBBC	Fast Back-to-Back Capable.	1	RO	RO
22	UDF	User Definable Features supported. 0 = not supported	0	RO	RO
21	66MHZ	66MHZ capable. Indicates if this device is capable of 66MHz operation. 1 = 66MHz capable.	1	RO	RW
20	CLI	Capabilities List Indicator, Not supported	0	RO	RO
19:10	-	reserved	00	RO	RO
9	FBBE	Fast Back-to-Back Enable. When set to a 1 enables the device to generate fast back-to-back cycles to different targets as a Master.	0	RW	RW
8	SER	System Error Enable. When set to a 1, enables the PCI_SERR_N output driver. 0 disables the driver.	0	RW	RW
7	SC	Stepping Control. When set to a 1, enables address stepping on the bus. This feature not supported.	0	RO	RO
6	PER	Parity Error Response. When set to a 1, enables reporting of parity errors on PCI_PERR_N. When set to 0, parity errors not reported on PCI_PERR_N but the DPE bit (bit 31) is still set.	0	RW	RW
5	PSE	Palette Snoop Enable. When set to a 1, enables VGA palette snooping. This feature not supported.	0	RO	RO



Register		pci_srcr (Sheet 2 of 2)			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
4	MWIE	Memory Write and Invalidate Enable. When set to a one, enables this device to generate the memory write and invalidate command.	0	RW	RW
3	SCE	Special Cycle Enable. When set, enables this device to monitor for Special Cycles. This feature not supported.	0	RO	RO
2	BME	Bus Master Enable. When set, enables this device to act as a bus Master.	0	RW	RW
1	MAE	Memory Access Enable. When set to a 1, enables memory accesses as a target.	0	RW	RW
0	IOAE	I/O Access Enable. When set to a 1, enables I/O accesses as a target.	0	RW	RW

10.5.2.3 Class Code /Revision ID Register

Register Name:						pci_ccrid																													
Block Base Address:						0xC00000						Offset Address						0x08						Reset Value						0x0b4000XY XY = pci_revision_id [7:0]					
Register Description:						Provides Class Code and Revision ID values as specified in the PCI 2.2 Local Bus Specification.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Class Code						Sub-Class Code						Interface						RevisionID																	

Register		pci_ccrid			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	Class Code	Class/Sub-Class identifier for the device as defined in the PCI specification. 0x0b = processor	0x0b	RO	RW
23:16	Sub-Class	Sub-Class identifier for Class Code 0x0b. 0x40 = coprocessor	0x40	RO	RW
15:8	Interface	Programming Interface code. Always 0x00 for this class.	0x00	RO	RW
7:0	RevisionID	Silicon A0 Revision for this device. Silicon A1 Revision for this device. Silicon A2 Revision for this device.	0x00 0x01 0x02	RO	RW



10.5.2.4 BIST/Header Type/Latency Timer/Cache Line Register

Register Name:						pci_bhlc																									
Block Base Address:		0xC00000						Offset Address						0x0c						Reset Value				0x00000000							
Register Description:						Provides BIST, Header Type, Latency Timer, and Cache Line Size registers as specified in the PCI 2.2 Local Bus Specification.																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIST						HeaderType						LatencyTimer						cache line													

Register		pci_bhlc			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	BIST	BIST control register, not supported	0x00	RO	RO
23	Header Type[7]	Single Function/Multi-Function Device. Set to 0 to identify this device as a single-function PCI device.	0	RO	RO
22:1 6	Header Type[6:0]	Configuration Header Type for this device. Set to 00	0x00	RO	RO
15:1 0	Latency Timer[7:2]	Latency Timer value in units of four PCI bus clocks.	0x00	RW	RW
9:8	Latency 1 Timer[1:0]	Hard-wired low order Latency Timer bits	0x0	RO	RO
7:0	Cache Line	Cache Line Size in units of 32-bit words.	0x00	RW	RW

10.5.2.5 Base Address 0 Register

Register Name:						pci_bar0																													
Block Base Address:						0xC00000						Offset Address						0x10						Reset Value						0x00000008					
Register Description:						PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
RWBase						FixedBase						FixedBase																		PREF	Type	MSI			

Register		pci_bar0 (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO



Register		pci_bar0 (Sheet 2 of 2)			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

10.5.2.6 Base Address 1 Register

Register Name:						pci_bar1																													
Block Base Address:						0xC00000						Offset Address						0x14						Reset Value						0x00000008					
Register Description:						PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
RWBase						FixedBase						FixedBase														PREF				Type		MSI			

Register		pci_bar1			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

10.5.2.7 Base Address 2 Register

Register Name:						pci_bar2																															
Block Base Address:						0xC00000						Offset Address						0x18						Reset Value						0x00000008							
Register Description:						PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.																				Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
RWBase						FixedBase						FixedBase														PREF		Type		MSI							



Register		pci_bar2			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

10.5.2.8 Base Address 3 Register

Register Name:						pci_bar3																													
Block Base Address:						0xC00000						Offset Address						0x1c						Reset Value						0x00000008					
Register Description:						PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
RWBase						FixedBase						FixedBase																		PREF	Type	MSI			

Register		pci_bar3			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

10.5.2.9 Base Address 4 Register

Register Name:						pci_bar4																									
Block Base Address:		0xC00000						Offset Address						0x20						Reset Value				0x00000008							
Register Description:						PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RWBase						FixedBase						FixedBase																PREF	Type		MSI



Register		pci_bar4			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

10.5.2.10 Base Address 5 Register

Register Name:					pci_bar5																										
Block Base Address:					0xC00000					Offset Address					0x24					Reset Value					0x00000001						
Register Description:					PCI Base Address register for AHB I/O space access. Format as specified in the PCI 2.2 Local Bus Specification.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RWBase																							FixedBase					(Rs v'd)		IO SI	

Register		pci_bar5			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	RWBase	Read/Write bits of Base Address register.	0x000000	RW	RW
7:2	FixedBase	Read-only bits of Base Address register. Specifies fixed 256 byte address range for this BAR.	0x00	RO	RO
1	reserved	Reserved	0	RO	RO
0	IOSI	I/O space indicator. Hard-wire to 1 for I/O space.	1	RO	RO

10.5.2.11 Subsystem ID/Subsystem Vendor ID Register

Register Name:						pci_sidsvid																													
Block Base Address:						0xC00000						Offset Address						0x2c						Reset Value						0x00000000					
Register Description:						Provides Subsystem Device ID and Susbsystem Vendor ID values as specified in the PCI 2.2 Local Bus Specification.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
SDeviceID																SVendorID																			



Register		pci_sidsvid			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	SDeviceID	Subsystem Device ID	0x0000	RO	RW
15:0	SVendorID	Subsystem Vendor ID	0x0000	RO	RW

10.5.2.12 Max_Lat, Min_gnt, Interrupt Pin, and Interrupt Line Register

Register Name:						pci_latent																													
Block Base Address:						0xC00000						Offset Address						0x3c						Reset Value						0x00000100					
Register Description:						Miscellaneous register provides Max Latency, Min Grant, Interrupt Pin and Interrupt Line parameters as specified in the PCI 2.2 Local Bus Specification.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
MaxLat						MinGnt						IntPin						IntLine																	

Register		pci_latent			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	MaxLat	Indicates how often this device needs access to the bus, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
23:16	MinGnt	Indicates the time interval required for a burst operation, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
15:8	IntPin	Indicates which interrupt pin this device connects to. Set to connect to INT_A#.	0x01	RO	RW
7:0	IntLine	Indicates interrupt line routing information.	0x00	RW	RW

10.5.2.13 Retry Timeout/trdy Timeout Register

Register Name:					pci_rtotto																										
Block Base Address:					0xC00000					Offset Address					0x40					Reset Value					0x00008080						
Register Description:					Specifies values for the Retry and TRDY timeout timers.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															RetryTO					TRDYTO											



Register		pci_rtotto			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	reserved	Reserved	0x00	RO	RO
15:8	RetryTO	Specifies value for the Retry timer. Specifies the maximum number of retries the Master Interface will accept before terminating the transaction. A value of 0 disables the timer and allows an unlimited number of Retry responses.	0x80	RW	RW
7:0	TRDYTO	Specifies value for the TRDY timer. Specifies the number of PCI clocks the Master Interface will wait before terminating a transfer with Master Abort when a target accepts a transaction by asserting PCI_DEVSEL_N but does not assert PCI_TRDY_N or PCI_STOP_N. A value of 0 disables the timer and the Master Interface will wait indefinitely for the Target to respond.	0x80	RW	RW

10.5.3 PCI Controller Configuration and Status Registers (CSRs)

These registers are accessible from the AHB bus and are memory mapped in the AHB address space. When the arbs_hsel_ppc_mmr is asserted, the lower AHB address bits select the accessed register. Registers are byte writable with individual bytes addressed according to the endianness setting of the AHB bus in pci_csr.ABE. Table 201 shows the address map for the CSRs. The AHB offset is relative to the base address for PCI Controller memory mapped registers in the AHB address space. The PCI offset is relative to the base address in pci_bar4 for accesses from the PCI bus. Byte addressing from PCI uses the PCI byte enable convention.

Table 201. CSR Address Map (Sheet 1 of 2)

AHB Offset	PCI Offset	Register Name	Description	Reset Value	Page
0x00	0x00	pci_np_ad	PCI non-prefetch address register	0x00000000	559
0x04	0x04	pci_np_cbe	PCI non-prefetch command/byte enables register	0x00000000	559
0x08	0x08	pci_np_wdata	PCI non-prefetch write data register	0x00000000	560
0x0c	0x0c	pci_np_rdata	PCI non-prefetch read data register	0x00000000	560
0x10	0x10	pci_crp_ad_cbe	PCI configuration port address/cmd/byte enables register	0x00000000	561
0x14	0x14	pci_crp_wdata	PCI configuration port write data register	0x00000000	562
0x18	0x18	pci_crp_rdata	PCI configuration port read data register	0x00000000	562
0x1c	0x1c	pci_csr	PCI Controller Control and Status register	0x0000000x	563
0x20	0x20	pci_isr	PCI Controller Interrupt Status register	0x00000000	564
0x24	0x24	pci_inten	PCI Controller Interrupt Enable register	0x00000000	565
0x28	0x28	pci_dmactrl	DMA control register	0x00000000	565
0x2c	0x2c	pci_ahbmembase	AHB Memory Base Address Register	0xc0000000	566
0x30	0x30	pci_ahbiobase	AHB I/O Base Address Register	0x00000000	567
0x34	0x34	pci_ahbmembase	PCI Memory Base Address Register	0x00000000	566
0x38	0x38	pci_ahbdoorbell	AHB Doorbell Register	0x00000000	568
0x3c	0x3c	pci_pcidoorbell	PCI Doorbell Register	0x00000000	569
0x40	0x40	pci_atpdma0_ahbaddr	AHB-to-PCI DMA AHB Address Register 0	0x00000000	569
0x44	0x44	pci_atpdma0_pciaddr	AHB-to-PCI DMA PCI Address Register 0	0x00000000	570
0x48	0x48	pci_atpdma0_length	AHB-to-PCI DMA Length Register 0	0x00000000	570



Table 201. CSR Address Map (Sheet 2 of 2)

AHB Offset	PCI Offset	Register Name	Description	Reset Value	Page
0x4c	0x4c	pci_atpdma1_ahbaddr	AHB-to-PCI DMA AHB Address Register 1	0x00000000	571
0x50	0x50	pci_atpdma1_pciaddr	AHB-to-PCI DMA PCI Address Register 1	0x00000000	571
0x54	0x54	pci_atpdma1_length	AHB-to-PCI DMA Length Register 1	0x00000000	572
0x58	0x58	pci_ptadma0_ahbaddr	PCI-to-AHB DMA AHB Address Register 0	0x00000000	572
0x5c	0x5c	pci_ptadma0_pciaddr	PCI-to-AHB DMA PCI Address Register 0	0x00000000	572
0x60	0x60	pci_ptadma0_length	PCI-to-AHB DMA Length Register 0	0x00000000	573
0x64	0x64	pci_ptadma1_ahbaddr	PCI-to-AHB DMA AHB Address Register 1	0x00000000	574
0x68	0x68	pci_ptadma1_pciaddr	PCI-to-AHB DMA PCI Address Register 1	0x00000000	574
0x6c	0x6c	pci_ptadma1_length	PCI-to-AHB DMA Length Register 1	0x00000000	575

10.5.3.1 PCI Controller Non-Prefetch Address Register

Register Name:						pci_np_ad																													
Block Base Address:						0xC00000						Offset Address						0x00						Reset Value						0x00000000					
Register Description:						PCI non-prefetch access address register. Provides address for CSR-initiated non-prefetch PCI accesses.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
np_address																																			

Register		pci_np_ad			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_address	Address of the non-prefetch PCI cycle. The format of the address depends on the PCI command type used.	0x00000000	none	RW

10.5.3.2 PCI Controller Non-Prefetch Command/Byte Enables Register

Register Name:					pci_np_cbe																											
Block Base Address:					0xC00000					Offset Address					0x04					Reset Value					0x00000000							
Register Description:					PCI non-prefetch access command/byte enables register. Provides PCI command and data byte enables for CSR-initiated non-prefetch PCI accesses.															Access:					(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(Reserved)																							NP_BE					NP_CMD				



Register		pci_np_cbe			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	reserved	reserved – read as 0	0x000000	none	RW
7:4	NP_BE	Byte enables driven onto the PCI_CBE_N[3:0] lines of the PCI bus during the data phase of the non-prefetch PCI access.	0x0	none	RW
3:0	NP_CMD	PCI command driven onto the PCI_CBE_N[3:0] lines of the PCI bus during the address phase of the non-prefetch PCI access.	0x0	none	RW

10.5.3.3 PCI Controller Non-Prefetch Write Data Register

Register Name:						pci_np_wdata																													
Block Base Address:						0xC00000						Offset Address						0x08						Reset Value						0x00000000					
Register Description:						PCI non-prefetch access write data register. Provides write data for CSR-initiated non-prefetch PCI write access.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
np_wdata																																			

Register		pci_np_wdata			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_wdata	Write data for the non-prefetch PCI write cycle.	0x00000000	none	RW

10.5.3.4 PCI Controller Non-Prefetch Read Data Register

Register Name:					pci_np_rdata																										
Block Base Address:		0xC00000					Offset Address					0x0c					Reset Value					0x00000000									
Register Description:						PCI non-prefetch access read data register. Holds read data from the CSR-initiated non-prefetch PCI read access.															Access:				(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
np_rdata																															

Register		pci_np_rdata			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_rdata	Read data from the non-prefetch PCI read cycle.	0x00000000	none	RO



10.5.3.5 PCI Controller Configuration Port Address/Command/Byte Enables Register

Register Name:						pci_crp_ad_cbe																									
Block Base Address:		0xC00000						Offset Address						0x10						Reset Value				0x00000000							
Register Description:						PCI configuration port address/command/byte enables register. Provides address, command, and data byte enables for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. A write to this register that sets the CRP_CMD[16] bit to a 0 (read) will initiate a read of the PCI Controller PCI configuration register addressed by CRP_AD[7:2]. The resultant read data will be written to the pci_crp_rdata register.														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)						CRP_BE				CRP_CMD				(Reserved)				CRP_AD													

Register		pci_crp_ad_cbe			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	reserved	reserved – read as 0	0x00	none	RO
23:2 0	CRP_BE	Active-low byte enables for a PCI configuration port write access. This field corresponds to byte lanes in the pci_crp_wdata register and addressed PCI configuration register as follows: CRP_BE[0] → bits 7:0 CRP_BE[1] → bits 15:8 CRP_BE[2] → bits 23:16 CRP_BE[3] → bits 31:24	0x0	none	RW
19:1 6	CRP_CMD	Command for the PCI configuration port access. xxx0 = read, xxx1 = write	0x0	none	RW
15:1 1	reserved	reserved – read as 0	00000	none	RO
10:0	CRP_AD	Byte address for the PCI configuration port access. PCI configuration registers are word-addressed so bits 1 and 0 should always be 0. Bits 10:8 specify the function number and must always be 000.	0x000	None	RW



10.5.3.6 PCI Controller Configuration Port Write Data Register

Register Name:						pci_crp_wdata																									
Block Base Address:						0xC00000				Offset Address				0x14				Reset Value				0x00000000									
Register Description:						PCI configuration port write data register. Provides write data for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. If the pci_crp_ad_cmd_be.CRP_CMD[16] bit is 1 (write), a write to pci_crp_wdata will initiate a write to the PCI Controller PCI configuration register addressed by pci_crp_ad_cmd_be.CRP_AD[7:2]. The pci_crp_ad_cmd_be.CRP_BE field determines which bytes are affected.														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRP_WDATA																															

Register		pci_crp_wdata			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	CRP_WDATA	Write data for the configuration port write access.	0x00000000	none	RW

10.5.3.7 PCI Controller Configuration Port Read Data Register

Register Name:						pci_crp_rdata																													
Block Base Address:						0xC00000						Offset Address						0x18						Reset Value						0x00000000					
Register Description:						PCI configuration port read data register. Provides read data for CSR-initiated read accesses of the PCI Controller PCI configuration registers in the PCI Core.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
CRP_RDATA																																			

Register		pci_crp_rdata			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	CRP_RDATA	Read data for the configuration port read access.	0x00000000	none	RO



10.5.3.8 PCI Controller Control and Status Register

Register Name:						pci_csr																													
Block Base Address:						0xC00000						Offset Address						0x1c						Reset Value						0x0000000x					
Register Description:						Control and status for the PCI Controller.																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)														PRST	IC	reserved						ASE	(Rsv'd)		DBT	ABE	PBS	ABS	ARBEN	HOST					

Register		pci_csr			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:17	reserved	reserved – read as 0	0x0000	RO	RO
16	PRST	PCI Reset. When set to a 1, resets the PCI controller and floats the outputs (even in the middle of a PCI transaction). Setting PRST does not park the bus.	0x0	RO	RW
15	IC	Initialization Complete. When at a logic 0 state, forces the PCI Controller Target Interface to retry PCI cycles. When set to a 1, PCI cycles will be accepted.	0	RO	RW
14:9	reserved	reserved – read as 0	0x00	RO	RO
8	ASE	Assert System Error. When set to a 1, the PCI_SERR_N output will be asserted for 1 PCI clock cycle if the pci_srcr.SER bit is set.	0	RO	RW
7:6	reserved	reserved – read as 0	00	RO	RO
5	DBT	Doorbell Test mode enable. When set to a 1, the doorbell registers pci_ahbdoorbell, pci_pcdoorbell become normal read/write registers from the AHB bus.	0	RO	RW
4	ABE	AHB big-endian addressing. When 0, little-endian addressing is employed on both AHB master and slave interfaces. When 1, big-endian addressing is implemented.	0	RO	RW
3	PBS	PCI byte swap. Controls byte lane data routing between PCI and AHB busses during PCI Target accesses of the AHB bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
2	ABS	AHB byte swap. Controls byte lane data routing between PCI and AHB busses during AHB Slave accesses of the PCI bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
1	ARBEN	Arbiter enable status. Indicates the state of the EX_ADDR[2] at the deassertion of RESET_IN_N. Note: Reset value is dependent on expansion bus strapping.	0 or 1	RO	RO
0	HOST	Host status. Indicates the state of the EX_ADDR[1] at the deassertion of RESET_IN_N. Note: Reset value is dependent on expansion bus strapping.	0 or 1	RO	RO



10.5.3.9 PCI Controller Interrupt Status Register

Register Name:						pci_isr																									
Block Base Address:						0xC00000				Offset Address				0x20						Reset Value				0x00000000							
Register Description:						Indicates the PCI controller interrupt source(s).														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																							PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE	

Register		pci_isr			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	reserved	reserved – read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt. Asserted high when any one of the bits in the pci_pcdoorbell register is set.	0	RO	RO
6	ADB	AHB Doorbell interrupt. Asserted high when any one of the bits in the pci_ahbdoorbell register is set.	0	RO	RO
5	PADC	PCI-to-AHB DMA Complete. Asserted high when a PCI-to-AHB DMA transfer is complete.	0	RO	RO
4	APDC	AHB-to-PCI DMA Complete. Asserted high when a AHB-to-PCI DMA transfer is complete.	0	RO	RO
3	AHBE	AHB Error indication. Set to a 1 when the AHB Master Interface receives an ERROR response.	0	RO	RW1C
2	PPE	PCI Parity Error. Set to a 1 when a parity error occurs on the PCI bus: Parity error detected during Master Interface read transaction. PCI_PERR_N asserted by an external target during a Master write transaction. Parity error detected during a Target write transaction.	0	RO	RW1C
1	PFE	PCI Fatal Error. Set to a 1 when one of the following errors occurs on the PCI bus: - Master abort (target did not respond) - Target abort - TRDY timeout (external target asserts PCI_DEVSEL_N but never asserts PCI_TRDY_N) - Retry timeout (external target issued more retries than specified by the RetryTO field in the pci_rtotto register)	0	RO	RW1C
0	PSE	PCI System Error. Set to a 1 when the PCI Controller detects that the PCI PCI_SERR_N signal has been asserted.	0	RO	RW1C



10.5.3.10 PCI Controller Interrupt Enable Register

Register Name:					pci_inten																										
Block Base Address:		0xC00000					Offset Address					0x24					Reset Value				0x00000000										
Register Description:						Interrupt enables for the interrupt status bits in the pci_isr register. Set to a 1 to enable the particular interrupt.														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																							PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE	

Register		pci_inten			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	reserved	reserved – read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt enable.	0	RO	RW
6	ADB	AHB Doorbell interrupt enable.	0	RO	RW
5	PADC	PCI-to-AHB DMA Complete interrupt enable.	0	RO	RW
4	APDC	AHB-to-PCI DMA Complete interrupt enable.	0	RO	RW
3	AHBE	AHB Error indication interrupt enable.	0	RO	RW
2	PPE	PCI Parity Error interrupt enable.	0	RO	RW
1	PFE	PCI Fatal Error interrupt enable.	0	RO	RW
0	PSE	PCI System Error interrupt enable.	0	RO	RW

10.5.3.11 DMA Control Register

Register Name:					pci_dmactrl																										
Block Base Address:					0xC00000					Offset Address					0x28					Reset Value					0x00000000						
Register Description:					Control and status for the DMA Controller channels.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																PADEN	PADC1	PADEN	(Reserved)			PADEN	APDE1	PADC1	APDE0	PADC0	(Rsv'd)				



Register		pci_dmactrl			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	reserved	reserved – read as 0	0x0000	RO	RO
15	PADE1	PCI-to-AHB DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC1 bit.	0	RO	RO
14	PADC1	PCI-to-AHB DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers is complete or terminated due to an error.	0	RO	RW1C
13	PADE0	PCI-to-AHB DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC0 bit.	0	RO	RO
12	PADC0	PCI-to-AHB DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers is complete or terminated due to an error.	0	RO	RW1C
11:9	reserved	reserved – read as 0	000	RO	RO
8	PADCEN	PCI-to-AHB DMA Complete interrupt enable.	0	RO	RW
7	APDE1	AHB-to-PCI DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC1 bit.	0	RO	RO
6	APDC1	AHB-to-PCI DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers is complete or terminated due to an error.	0	RO	RW1C
5	APDE0	AHB-to-PCI DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC0 bit.	0	RO	RO
4	APDC0	AHB-to-PCI DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers is complete or terminated due to an error.	0	RO	RW1C
3:1	reserved	reserved – read as 0	000	RO	RO
0	APDCEN	AHB-to-PCI DMA Complete interrupt enable.	0	RO	RW

10.5.3.12 AHB Memory Base Address Register

Register Name:						pci_ahbmembase																									
Block Base Address:		0xC00000				Offset Address				0x2c				Reset Value				0xc0000000													
Register Description:						Provides upper 8 AHB address bits for PCI accesses of AHB bus. Lower 24 bits of AHB address provided directly from PCI bus. Four AHBbase fields correspond to accesses from the PCI bus that target addresses in PCI configuration base address registers pci_bar0/1/2/3.														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHBbase0						AHBbase1						AHBbase2						AHBbase3													



Register		pci_ahbmembase			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	AHBbase0	Upper 8 AHB address bits for PCI accesses that target pci_bar0. By default this register maps to an upper region of the AHB memory map.	0xc0	RO	RW
23:1 6	AHBbase1	Upper 8 AHB address bits for PCI accesses that target pci_bar1.	0x00	RO	RW
15:8	AHBbase2	Upper 8 AHB address bits for PCI accesses that target pci_bar2.	0x00	RO	RW
7:0	AHBbase3	Upper 8 AHB address bits for PCI accesses that target pci_bar3.	0x00	RO	RW

10.5.3.13 AHB I/O Base Address Register

Register Name:						pci_ahbiobase																													
Block Base Address:						0xC00000						Offset Address						0x30						Reset Value						0x00000000					
Register Description:						Provides upper 24 AHB address bits for PCI accesses of AHB I/O space. Lower 8 bits of AHB address provided directly from PCI bus.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)						IObase																													

Register		pci_ahbiobase			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2 4	reserved	Reserved – read as 0	0x00	RO	RO
23:0	IObase	Upper 24 AHB address bits for PCI accesses that target pci_bar5.	0x000000	RO	RW

10.5.3.14 PCI Memory Base Address Register

Register Name:					pci_pcimembase																										
Block Base Address:		0xC00000					Offset Address					0x34					Reset Value					0x00000000									
Register Description:					Provides upper 8 PCI address bits for AHB accesses of PCI memory space. Lower 24 bits of PCI address provided directly from the AHB bus. Four Membase fields correspond to accesses from the AHB bus that target specific AHB address ranges.															Access:			(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Membase0						Membase1						Membase2						Membase3													



Register		pci_pcimembase			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	PClbase0	Upper 8 PCI address bits for AHB accesses that target the first 16MB PCI memory partition.	0x00	RO	RW
23:16	PClbase1	Upper 8 PCI address bits for AHB accesses that target the second 16MB PCI memory partition.	0x00	RO	RW
15:8	PClbase2	Upper 8 PCI address bits for AHB accesses that target the third 16MB PCI memory partition.	0x00	RO	RW
7:0	PClbase3	Upper 8 PCI address bits for AHB accesses that target the fourth 16MB PCI memory partition.	0x00	RO	RW

10.5.3.15 AHB Doorbell Register

Register Name:						pci_ahbdoorbell																													
Block Base Address:						0xC00000						Offset Address						0x38						Reset Value						0x00000000					
Register Description:						This register is write-1-to-set from PCI and write-1-to-clear from AHB. The PCI device writes a 1 to a bit or pattern of bits to generate the interrupt. The AHB agent reads the register and writes 1(s) to clear the bit(s) and deassert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
ADB																																			

Register		pci_ahbdoorbell			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	ADB	PCI generated doorbell interrupt to an AHB agent. Normally read/write-1-to-set from PCI and read/write-1-to-clear from AHB. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x00000000	RW1S	RW1C (RW if pci_csr.DBT=1)



10.5.3.16 PCI Doorbell Register

Register Name:		pci_pcidoorbell																																			
Block Base Address:	0xC00000					Offset Address					0x3c					Reset Value					0x00000000																
Register Description:						The Intel XScale processor writes this register to generate an interrupt to an external PCI device on PCI_INTA_N. Any bit set to a 1 will generate the PCI interrupt if the PCI doorbell interrupt is enabled (pci_inten.PDBEN = 1). This register is write-1-to-set from AHB and write-1-to-clear from PCI. The Intel XScale processor writes a 1 to a bit or pattern of bits to generate the interrupt. The external PCI device reads the register and writes 1(s) to clear the bit(s) and deassert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.																			Access:				(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
PDB																																					

Register		pci_pcidoorbell				
Bits	Name	Description	Reset Value	PCI Access	AHB Access	
31:0	PDB	AHB generated doorbell interrupt to PCI. Normally read/write-1-to-set from AHB and read/write-1-to-clear from PCI. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x00000000	RW1C	RW1S (RW if pci_csr.DBT=1)	

10.5.3.17 AHB-to-PCI DMA AHB Address Register 0

Register Name:					pci_atpdma0_ahbaddr																										
Block Base Address:					0xC00000					Offset Address					0x40					Reset Value					0x00000000						
Register Description:					Source address on the AHB bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_ahbaddr to allow buffering of DMA transfer requests.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
address																														0	0

Register		pci_atpdma0_ahbaddr			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

**10.5.3.18 AHB-to-PCI DMA PCI Address Register 0**

Register Name:						pci_atpdma0_pciaddr																													
Block Base Address:						0xC00000						Offset Address						0x44						Reset Value						0x00000000					
Register Description:						Destination address on the PCI bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_pciaddr to allow buffering of DMA transfer requests.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
address																																0	0		

Register		pci_atpdma0_pciaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access	
31:2	address	PCI word address	0x00000000	RO	RW	
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO	

10.5.3.19 AHB-to-PCI DMA Length Register 0

Register Name:				pci_atpdma0_length																											
Block Base Address:				0xC00000				Offset Address				0x48				Reset Value				0x00000000											
Register Description:				Provides word count and control for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_length to allow buffering of DMA transfer requests.																				Access:				(See below.)			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN		(Rsvd)		DS		(Reserved)										Wordcount															

Register		pci_atpdma0_length				
Bits	Name	Description	Reset Value	PCI Access	AHB Access	
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW	
30:29	reserved	Reserved. Read as 0.	00	RO	RO	
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW	
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO	
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW	



10.5.3.20 AHB-to-PCI DMA AHB Address Register 1

Register Name:						pci_atpdma1_ahbaddr																													
Block Base Address:						0xC00000						Offset Address						0x4c						Reset Value						0x00000000					
Register Description:						Source address on the AHB bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_ahbaddr to allow buffering of DMA transfer requests.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
address																															0	0			

Register		pci_atpdma1_ahbaddr															
Bits	Name	Description										Reset Value	PCI Access	AHB Access			
31:2	address	AHB word address										0x00000000	RO	RW			
1:0		Lower AHB address bits hard-wired to zero.										00	RO	RO			

10.5.3.21 AHB-to-PCI DMA PCI Address Register 1

Register Name:					pci_atpdma1_pciaddr																											
Block Base Address:					0xC00000					Offset Address					0x50					Reset Value					0x00000000							
Register Description:					Destination address on the PCI bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_pciaddr to allow buffering of DMA transfer requests.															Access:					(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
address																															0	0

Register		pci_atpdma1_pciaddr															
Bits	Name	Description										Reset Value	PCI Access	AHB Access			
31:2	address	PCI word address										0x00000000	RO	RW			
1:0		Lower PCI address bits hard-wired to zero.										00	RO	RO			



10.5.3.22 AHB-to-PCI DMA Length Register 1

Register Name:					pci_atpdma1_length																											
Block Base Address:					0xC00000					Offset Address					0x54					Reset Value					0x00000000							
Register Description:					Provides word count and control for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_length to allow buffering of DMA transfer requests.																	Access:					(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
EN		(Rsvd)		DS		(Reserved)										wordcount																

Register		pci_atpdma1_length				
Bits	Name	Description	Reset Value	PCI Access	AHB Access	
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW	
30:29	reserved	Reserved. Read as 0.	00	RO	RO	
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW	
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO	
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW	

10.5.3.23 PCI-to-AHB DMA AHB Address Register 0

Register Name:						pci_ptadma0_ahbaddr																													
Block Base Address:						0xC00000						Offset Address						0x58						Reset Value						0x00000000					
Register Description:						Destination address on the AHB bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_ahbaddr to allow buffering of DMA transfer requests.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
address																																0	0		

Register		pci_ptadma0_ahbaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access	
31:2	address	AHB word address	0x00000000	RO	RW	
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO	



10.5.3.24 PCI-to-AHB DMA PCI Address Register 0

Register Name:						pci_ptadma0_pciaddr																											
Block Base Address:		0xC00000						Offset Address						0x5c						Reset Value				0x00000000									
Register Description:						Source address on the PCI bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_pciaddr to allow buffering of DMA transfer requests.																		Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
address																																0	0

Register		pci_ptadma0_pciaddr															
Bits	Name	Description											Reset Value	PCI Access	AHB Access		
31:2	address	PCI word address											0x00000000	RO	RW		
1:0		Lower PCI address bits hard-wired to zero.											00	RO	RO		

10.5.3.25 PCI-to-AHB DMA Length Register 0

Register Name:						pci_ptadma0_length																										
Block Base Address:			0xC00000						Offset Address						0x60						Reset Value				0x00000000							
Register Description:						Provides word count and control for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_length to allow buffering of DMA transfer requests.																Access:				(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
EN		(Rsvd)		DS		(Reserved)											wordcount															

Register		pci_ptadma0_length															
Bits	Name	Description											Reset Value	PCI Access	AHB Access		
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.											0	RO	RW		
30:29	reserved	Reserved. Read as 0.											00	RO	RO		
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.											0	RO	RW		
27:16	reserved	Reserved. Read as 0.											0x000	RO	RO		
15:0	wordcount	Number of words to transfer.											0x0000	RO	RW		



10.5.3.26 PCI-to-AHB DMA AHB Address Register 1

Register Name:						pci_ptadma1_ahbaddr																													
Block Base Address:						0xC00000						Offset Address						0x64						Reset Value						0x00000000					
Register Description:						Destination address on the AHB bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_ahbaddr to allow buffering of DMA transfer requests.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
address																																0	0		

Register		pci_ptadma1_ahbaddr										Reset Value	PCI Access	AHB Access
Bits	Name	Description												
31:2	address	AHB word address										0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.										00	RO	RO

10.5.3.27 PCI-to-AHB DMA PCI Address Register 1

Register Name:					pci_ptadma1_pciaddr																												
Block Base Address:					0xC00000					Offset Address					0x68					Reset Value					0x00000000								
Register Description:					Source address on the PCI bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_pciaddr to allow buffering of DMA transfer requests.															Access:					(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
address																																0	0

Register		pci_ptadma1_pciaddr										Reset Value	PCI Access	AHB Access
Bits	Name	Description												
31:2	address	PCI word address										0x00000000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.										00	RO	RO



10.5.3.28 PCI-to-AHB DMA Length Register 1

Register Name:					pci_ptadma1_length																											
Block Base Address:					0xC00000					Offset Address					0x6c					Reset Value					0x00000000							
Register Description:					Provides word count and control for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_length to allow buffering of DMA transfer requests.															Access:					(See below.)							
31	30	29	28	27	26	25	24		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN		(Rsvd)		BE		(Reserved)										wordcount																

Register		pci_ptadma1_length			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29	reserved	Reserved. Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.	0	RO	RW
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

10.6 Error / Abnormal Conditions

This section describes the behavior of the PCI Controller under various error conditions that can occur on the PCI and AHB busses.

10.6.1 Error Handling as a PCI Target

10.6.1.0.1 A PCI Target Read Received an Error Response During the AHB Read Operation and the PCI Transfer is Not Complete

This case occurs when the Target Interface is delivering data to the PCI Initiator and an abort indication is received from the AHB Master Interface due to an ERROR response received during the AHB read transfer.

1. The AHB Master terminates the read transfer and stops delivering data to the PCI Core.
2. The PCI Target delivers all remaining data in the Target Read FIFO to the PCI Initiator then terminates the cycle with a Target Abort response. With the ECC/parity implementation of the DDR1 and Expansion bus controller on the IXP45X/IXP46X network processors, if an ECC or parity error exists on any word within an 8-word line (even if the PCI Initiator does not ask for this word), the PCI will respond with Target Abort on the first word of the read transfer.
3. The PCI Configuration Register pci_srcr.STA bit is set to indicate that this device signalled a Target Abort.
4. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.



10.6.1.0.2 A PCI Target Read Received an Error Response During the AHB Read Operation After the PCI Transfer is Complete

This scenario is possible since the AHB Master Interface does not know ahead of time how much data the PCI Initiator will need and thus keeps generating AHB read cycles until the Target Interface detects the end of the PCI read cycle. Due to latencies involved in notifying the AHB Master Interface, additional AHB reads will be generated after the read cycle completes on PCI. If an error response is received during one of these read-ahead AHB cycles:

1. The PCI cycle completes normally with all requested data delivered to the Initiator and no Target Abort response generated.
2. Any additional data left in the Target Read FIFO is discarded.
3. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.

10.6.1.0.3 A PCI Target Write Received an Error Response During the AHB Write Operation

1. The AHB Master Interface terminates the write transfer.
2. Any additional data in the Target Write FIFO is read out and discarded until the last-data marker is encountered indicating that the PCI cycle is complete.
3. Since writes are posted by the PCI Target Interface, the cycle could complete on PCI well before the AHB write error happens. For this reason, a Target Abort response is not generated on the PCI bus. All writes complete normally on PCI regardless of the status of the transfer on the AHB bus.
4. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.

10.6.1.0.4 The PCI Target Interface Detected an Address Phase Parity Error During a Target Write

1. If the transfer is longer than 1 data phase, the PCI Target Interface terminates the cycle with a Target Abort and sets the PCI configuration Register bit pci_srcr.STA
2. If the transfer consists of just one data phase, the PCI Target Interface accepts the transfer without signalling Target Abort but the data is discarded and no AHB write operation is performed.
3. The PCI Configuration Register bit pci_srcr.DPE is set indicating that a parity error has been detected.
4. If the PCI Configuration Register bit pci_srcr.SER is set, the PCI_SERR_N signal is asserted on the bus and the pci_srcr.SSE bit is set indicating that a system error has been signalled by this device.

10.6.1.0.5 The PCI Target Interface Detected a Data Phase Parity Error During a Target Write

1. The PCI Target Interface tags the errored data but otherwise processes the transfer normally, passing the data to the AHB Master.
2. The PCI Configuration Register bit pci_srcr.DPE is set to a 1. If the pci_srcr.PER bit is set, the PCI_PERR_N signal is asserted on the PCI bus.
3. The AHB Master Interface ignores the error and processes the write transfer normally.

10.6.1.0.6 The Initiator of a PCI Target Read Asserts PCI_PERR_N During the Transfer

1. The error indication is ignored. Handling of the error is left to the initiator of the transfer.



10.6.2 Error Handling as a PCI Initiator During PCI Direct Access from the AHB Bus

This section describes error handling procedures when the PCI Initiator Interface encounters a fatal error condition during a PCI transfer request received from the AHB Target Interface.

10.6.2.0.1 An AHB Target Read Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Read Operation

Note: A PCI_TRDY_N timeout occurs if the intended PCI target of the transfer asserts PCI_DEVSEL_N but fails to assert PCI_TRDY_N or abort the transfer with a Target Abort within the number of PCI clocks specified in pci_rtotto.TRDYTO. A RETRY Timeout occurs if the intended target exceeds the number of RETRY responses indicated in pci_rtotto.RetryTO.

1. If the AHB read operation is not complete, the AHB Target Interface asserts an ERROR response to terminate the AHB operation. If the AHB read is complete because the PCI error occurred during a read-ahead transfer, no action is taken. In either case, the AHB Target Interface blocks any further PCI requests until the error condition has cleared.
2. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
3. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

10.6.2.0.2 An AHB Target Write Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Write Operation

1. The Initiator Request FIFO is flushed and all queued requests are lost. Any data in the Initiator Transmit FIFO is flushed as well.
2. If the write is still in progress on the AHB bus, an ERROR response is not issued. The remaining data in the burst is discarded and the AHB transfer completes normally.
3. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
4. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

Caution: Note that in this error condition, the entire contents of the Initiator Request FIFO is discarded. Any queued write operations will be lost since they have already been posted by the AHB Target Interface. A queued read, though flushed from the queue, will complete since the AHB Master that originated the read must try the transfer again after getting a RETRY response. After the error condition has cleared, the AHB Target Interface will request the read again when the master retries the operation.

10.6.2.0.3 The PCI Initiator Interface Detected a Parity Error During a PCI Read Operation

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.
3. The PCI Configuration Register bit pci_srcr.DPE is set.

10.6.2.0.4 The Target of a PCI Write Operation Asserted PCI_PERR_N During the Transfer

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.



10.6.3 Error Handling as a PCI Initiator During Non-Prefetch Operations

This section describes error handling procedures when the PCI Initiator Interface encounters a fatal error condition during a PCI transfer request initiated by a non-prefetch operation.

10.6.3.0.1 The Non-Prefetch Operation Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Operation

1. The non-prefetch operation terminates and if the operation is a read, no data is written to the pci_np_rdata register. If the operation is a write, the Initiator Request FIFO and Initiator Transmit FIFO are flushed but since requests cannot be queued behind a non-prefetch operation, no requests can occur.
2. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
3. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

10.6.3.0.2 The PCI Initiator Interface Detected a Parity Error During a PCI Read Operation

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.
3. The PCI Configuration Register bit pci_srcr.DPE is set.

10.6.3.0.3 The Target of a PCI Write Operation Asserted PCI_PERR_N During the Transfer

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.

10.6.4 Error Handling During PCI-to-AHB DMA Channel Operations

10.6.4.0.1 A PCI Read Received a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.PADCx and pci_dmactrl.PADEx (x = 0 or 1 depending on which DMA buffer set encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.PADCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and will commence normally.
4. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.

10.6.4.0.2 An AHB Write Received an Error Response During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.PADCx and pci_dmactrl.PADEx (x = 0 or 1 depending on which DMA buffer set encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.PADCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and will commence normally.
4. The pci_isr.AHBE bit is set to indicate a fatal PCI error has occurred.



10.6.5 Error Handling During AHB-to-PCI DMA Channel Operations

10.6.5.0.1 A PCI Write Received a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.APDCx and pci_dmactrl.APDEx (x = 0 or 1 depending on which DMA buffer set encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.APDCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and will commence normally.
4. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.

10.6.5.0.2 An AHB Read Received an Error Response During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.APDCx and pci_dmactrl.APDEx (x = 0 or 1 depending on which DMA buffer set encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.APDCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and will commence normally.
4. The pci_isr.AHBE bit is set to indicate a fatal PCI error has occurred.

§ §





11.0 Memory Controller

This chapter describes the integrated Memory Controller Unit (DDRI MCU) of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The operating modes, initialization, external interfaces, and implementation are detailed in this chapter.

11.1 Overview

The IXP45X/IXP46X network processors integrate a high performance, multi-ported Memory Controller to provide a direct interface between the processor and its local memory subsystem. The Memory Controller supports:

- 128/256/512-Mbit, 1-Gbit DDRI SDRAM technology support.
- Unbuffered DRAM support only, no registered DRAM support.
- Dedicated port for Intel XScale® Processor to DDRI SDRAM (Supports critical word first reads).
- Between 32 Mbytes and 1 Gbytes of 32-bit DDRI SDRAM for low cost solutions.
- Two AHB ports for access from units other than Intel XScale processor (no critical word first support).
- All MMR accesses must go through the South AHB port.
- Single-bit error correction, multi-bit detection support (ECC).
- 32-, 40- wide Memory Interfaces (non-ECC and ECC support).

The DDRI SDRAM interface provides a direct connection to a reliable, high bandwidth memory subsystem. The DDRI SDRAM interface consists of a 32-bit wide data path to support up to 1066 Mbytes/sec. throughput. An 8-bit Error Correction Code (ECC) across each 32-bit word improves system reliability. The ECC is stored into the DDRI SDRAM array along with the data and is checked when the data is read. If the code is incorrect, the MCU corrects the data (if possible) before reaching the initiator of the read. User-defined fault correction software is responsible for scrubbing the memory array.

The MCU supports two physical banks of DDRI SDRAM in the form of discrete chips.

— The MCU only supports Unbuffered DDRI memory types.

- The MCU supports a 32-bit DDRI SDRAM data interface.
- The MCU contains transaction queues for each port enabling pipelining of transactions to the DDRI SDRAM for maximum performance.
- The MCU provides two chip enables to the memory subsystem. These two chip enables service the DDRI SDRAM subsystem (one per bank).
- Performance monitors for page hit / miss and read latency per port.



11.2 Theory of Operation

The memory controller of the IXP45X/IXP46X network processors translates the internal bus and core processor transactions into the protocol supported by the DDR1 SDRAM memory subsystem.

11.2.1 Functional Blocks

The following needs to be considered when creating low level software for the IXP45X/IXP46X network processors.

The Memory Controller Unit (MCU) is made up of 3 parts (see [Figure 101](#)). The MCU core which connects via MPI (Memory Port Interfaces) to the MAB (MPI to AHB bridge) and the Bus Interface Unit (BIU). In turn the MAB connects the MCU to the North and South AHB. Where the BIU connects the Intel XScale processor to the MCU MPI port and the South AHB.

The MAB supports posting of 1 write from the North AHB and 1 write from the South AHB. This means that once a master on the North and South AHB can perform a write to the MAB and the MAB will accept the write and free the AHB bus allowing the AHB master to continue. Shortly after the MAB accepts a write transaction, the MAB will send the request to the MCU core which in turn will perform a write to DDR.

The BIU also contains write posting capabilities. Because the BIU and MAB contain write posting capabilities, there could exist a race condition for write transactions posted in the BIU and MAB targeted to the same address.

There is no enforcement of write ordering between the North AHB and South AHB in the MAB for write requests that arrive at nearly the same time that are destined for the same exact DDR memory location. Therefore if an NPE and a South AHB master (PCI, Expansion Bus, USB or Intel XScale processor with MPI port disabled) write to the same exact DDR memory location at the same time there is no guarantee which write will reach memory last.

Furthermore there is no enforcement of write ordering for writes posted to the MAB slightly before the Intel XScale processor (with MPI enabled) writes to the same exact DDR memory location. In order to ensure a MAB write completes before the Intel XScale processor (with MPI enabled) writes to the same exact DDR memory location, the Intel XScale processor must perform a read of that location before performing a write.

As stated above both masters must be performing writes to the same exact DDR memory location at nearly the same time. There is no problem with one master reading and one master writing the same exact DDR memory location at the same time.

Example:

Masters:

- Master on the Expansion Bus
- Intel XScale processor

Transactions:

1. Master on the expansion bus performs a write (destined for DDR) to the expansion bus controller's internal queue.
2. Expansion bus controller initiates a South AHB write to the MAB, which is accepted in the South AHB posted write queue.



3. The MAB some time later sends the write request to the MCU core which performs the write to DDR memory location A2 (any DDR memory location).
4. Master on the expansion bus writes a pointer to the just written data (destined for Advanced Queue Manager) to the expansion bus controller's internal queue.
5. Expansion bus controller initiates a South AHB write to the Advanced Queue Manager.
6. Intel XScale processor reads pointer from Advanced Queue Manager
7. Intel XScale processor writes a timestamp to location A2.

In the above example the Expansion Bus write may complete before or after the Intel XScale processor write.

In this example the Advanced Queue Manager is not required to have this race condition. The same condition can be observed using an interrupt to hand control from an AHB master to the Intel XScale processor.

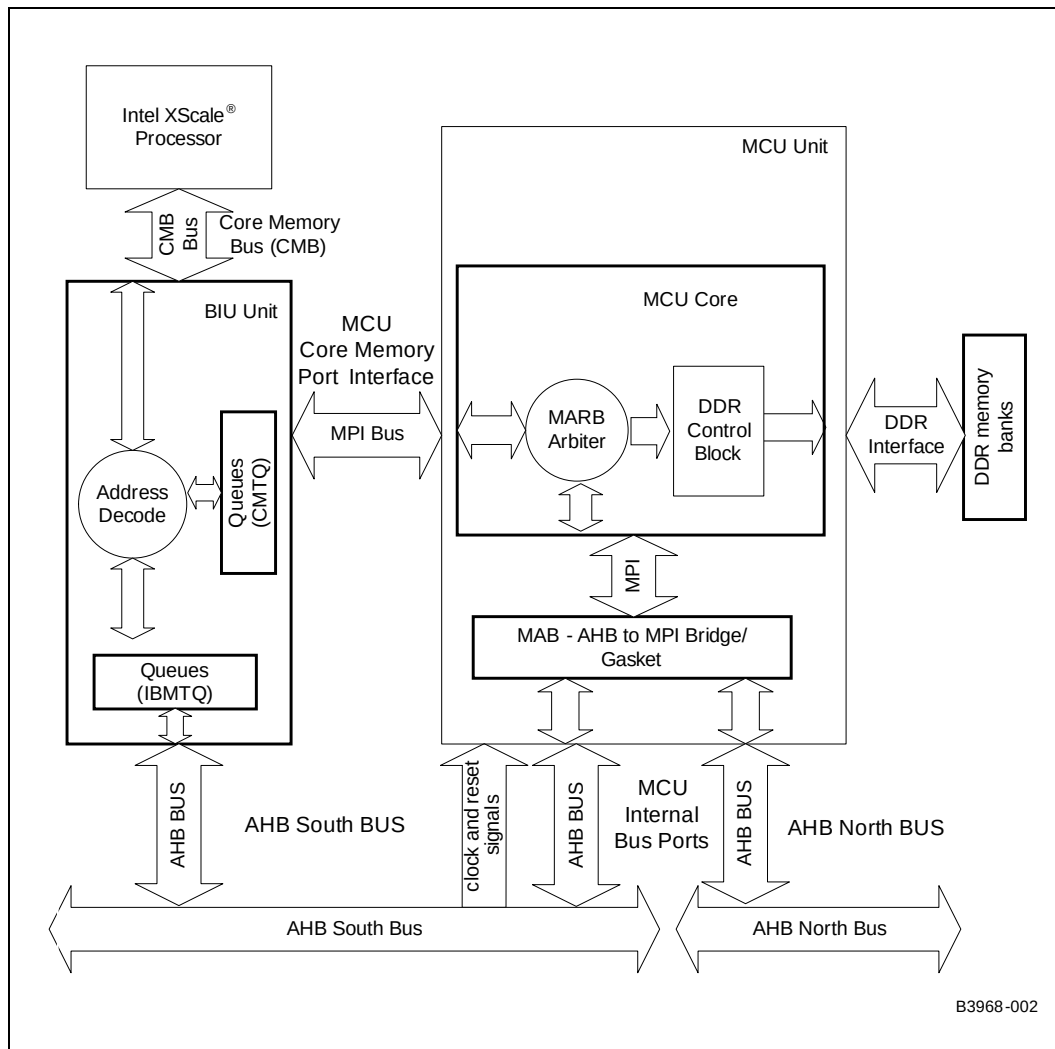
How to avoid

Any of the following will eliminate the race condition between an AHB master writing to the same exact DDR memory location that the Intel XScale processor is writing to.

1. Intel XScale processor needs to perform a read before it tries to write to the same exact DDR memory location. As stated above the MCU core will ensure any writes in the MAB or BIU will be processed before the read is returned thus ensuring the newest data is in DDR.
2. The AHB master performs a read of the location before handing control over to the Intel XScale processor.
3. The AHB master performs a write to a different location before handing control over to the Intel XScale processor.
4. Disable the BIU MPI port.

No consideration is required if two or more masters write to adjacent memory locations because each write is destined to it's own memory location and if a read is preformed the MCU core will ensure the newest data is returned.

Figure 101. Memory Controller Block Diagram



11.2.1.1 Transaction Ports

The MCU provides three transaction ports for DDR1 SDRAM access. They consist of two Internal Bus ports (AHB ports for the north and south AHB buses) and a direct port from the BIU (Core Processor Port):

11.2.1.1.1 Core Processor Port (From BIU)

The Core Processor Port provides a direct connection between the core bus interface of the IXP45X/IXP46X network processors and the Memory Controller. This Core Processor Port allows core transactions targeting the DDR1 SDRAM to pass directly to the DDR1 SDRAM.



11.2.1.1.2 Internal Bus Ports (North and South AHB)

The two Internal Bus Ports (North AHB and South AHB) provide the connection to the DDR1 SDRAM from the Internal AHB Buses. All peripheral unit transactions targeting the DDR1 SDRAM are claimed by these ports. Also all accesses to MMR space will be through the south AHB Internal bus port, this includes Intel XScale processor MMR accesses.

11.2.1.2 Address Decode Blocks

Address Decode is performed for transactions from input ports to determine if the MCU should claim the transaction. There are two address ranges the MCU ports can claim transactions: DDR1 SDRAM memory space and Memory-Mapped Register (MMR) space.

11.2.1.2.1 DDR1 SDRAM Memory Space

The DDR1 SDRAM memory space is defined as 0x0000_0000 to 0x3FFF_FFFF (1GB) for the IXP45X/IXP46X network processors.

Note: At boot time, the expansion bus resides at location 0000_0000h instead of the DDR1 SDRAM. The DDR1 controller must be configured first and the expansion bus decode programmed to sit at its higher address before DDR1 transactions can be initiated. See [Section 12.4.1.1, “Expansion Bus Address Space”](#) for details.

11.2.1.2.2 Memory-Mapped Register Space

The MCU MMR memory space is CCFF E500H to CCFF E5FFH and CCFF F500H to CCFF F5FFH. See [Section 11.6, “Register Definitions”](#) for details.

The Address Decode for each port is based on the same registers which are only accessible from the South AHB internal bus.

Read and write accesses to all Configuration/Status registers (CSR) must be single word transfers. Byte, half-word, burst, or coalesced transfers are not supported and result in unpredictable operation. All CSR reserved bits must be written with zero.

Each port decodes inbound transactions for these address ranges. The Address Decode blocks determine how the Memory Controller responds to inbound transactions. The details of the address decode for each port is described below.

11.2.1.2.3 Core Processor Port Address Decode

The address decode block for the Core Processor transactions resides in the BIU. The Core MCU port therefore does not require any decode, and will process any transaction received from the BIU via the Core MCU Port.

If the Core Processor Memory Transaction Queue is disabled (refer to [Section 12.5.10, “Configuration Register 1”](#) bit 31 MPI_EN) all core transactions are directed to the Core Internal Bus Transaction Queue of the BIU, and would not be sent to the Core MCU port (refer to [Figure 101 on page 584](#)). Therefore core transactions which address the DDR1 SDRAM, will be claimed by the IB port of the MCU and processed via that port as any other IB transaction. It is recommended that MCU transaction queues are not disabled.

11.2.1.2.4 Internal Bus Port Address Decode

Internal Bus transactions are decoded to determine if they address the DDR1 SDRAM Memory Space or MCU MMR Space. If the transaction addresses either of these two spaces, the transaction is claimed by one of the Internal Bus Ports. (Only South AHB port can claim MMR accesses.) If the transaction addresses the DDR1 SDRAM Memory



Space, the transaction is queued in the Internal Bus Port Transaction Queue. If the transaction addresses the MCU MMR Space, the transaction is serviced by accessing the Configuration Register block.

11.2.1.3 Memory Transaction Queues

There are two transaction queues for transactions which address the MCU DDRI Memory Space. One transaction queue is for Core Processor transactions and one transaction queue is for Internal Bus transactions (North and South AHB).

11.2.1.3.1 Core Processor Memory Transaction Queue (CMTQ)

The CMTQ stores memory transactions from the core which have not been processed by the Memory Controller. The CMTQ supports 8 Core Processor posted write transactions up to 16-Bytes each. The CMTQ also supports eight Core Processor read transactions up to 32 Bytes each, which equals the maximum number of outstanding transactions the Intel XScale® Processor Processor Bus Controller can support. The eight read transactions are:

- core DCU: 4 load requests to unique cachelines
- IFU: 2 - prefetch
- IMM: 1 - tablewalk
- DMM: 1 - tablewalk

11.2.1.3.2 AHB Internal Bus Memory Transaction Queue (IBMTQ)

Each IBMTQ stores memory transactions from the Internal Buses that address the DDRI SDRAM Memory Space. Each IBMTQ can hold 2 outstanding Internal Bus read or write transaction requests. The read transactions are processed one at a time, each with up to 32-Bytes of read data. The IBMTQ also supports 2 posted write transactions up to 32 Bytes each.

Note: The MCU Internal Bus Port will not split or retry Internal Bus Read Transactions. Instead it will stay on the bus until the read data is returned from the DRAM. This enables the queuing of outstanding read transactions without modifying the existing peripheral units of the IXP45X/IXP46X network processors. Internal Bus Write Transactions will be posted to improve bus bandwidth.

11.2.1.4 Configuration Registers

The Configuration Registers block contains all of the memory-mapped registers listed in [Section 11.6, “Register Definitions”](#). These registers define the memory subsystem connected to the IXP45X/IXP46X network processors. The status registers indicate the current MCU status.

11.2.1.5 Refresh Counter

The Refresh Counter block keeps track of when the DDRI SDRAM devices need to be refreshed. The refresh interval is programmed in the [Section 11.6.14, “Refresh Frequency Register RFR”](#). Once the 12-bit refresh counter reaches the programmed interval, the DDRI SDRAM state machine issues a refresh command to the DDRI SDRAM devices. If a transaction is currently in progress, the DDRI SDRAM State Machine waits for the completion of the transaction to issue the refresh cycle. See [Section 11.2.2.12, “DDRI SDRAM Refresh Cycle”](#) for more details.

Note: If the memory interface is busy when the refresh counter expires, it is possible for the MCU to generate more than one refresh cycle when the memory interface becomes available.



Note: If the memory controller is completing transactions from the core processor memory transaction queue (CMTQ) when the refresh counter expires, the Memory Controller Arbiter (MARB) will complete the CMTQ tenure based on the [Section 11.6.12, “MCU Port Transaction Count Register MPTCR”](#), before any refresh cycles are issued. Once refresh cycles are initiated, all queued refresh transactions are to be completed in a single tenure.

11.2.1.6 DDRI SDRAM Control Block

The DDRI SDRAM Control Block contains all functionality to process the DDRI SDRAM data accesses per the transactions issued by the MARB. To process a transaction the DDRI SDRAM Control Block employs several sub-blocks. The sub-blocks include the Page Control block, DDRI SDRAM State Machine and Pipeline Queues, and Error Correction Logic.

11.2.1.6.1 Page Control Block

The Page Control Block records and maintains the open DDRI SDRAM pages. The MCU can keep a maximum of eight pages open simultaneously (four per bank). This block keeps track of open pages and determines if the transactions hit an open page. See [Section 11.2.2.6, “Page Hit/Miss Determination”](#) for more details.

11.2.1.6.2 DDRI SDRAM State Machine and Pipeline Queues

Since the MCU generates error correction codes based on the data, the MCU is a pipelined architecture. Pipelining also ensures acceptable AC timings to the memory interfaces. The DDRI SDRAM state machine pipelines DDRI SDRAM memory operations for several clocks.

11.2.1.6.3 Error Correction Logic

The Error Correction Logic generates the ECC code for DDRI SDRAM reads and writes. For reads, this logic compares the ECC codes read with the locally generated ECC code. If the codes mismatch, then the Error Correction Logic determines the error type. For a single-bit error, this block determines which bit is in error and corrects the error. For a single-bit or multi-bit error, the Error Correction Logic logs the error in ELOG0 and ELOG1. See [Section 11.2.3, “Error Correction and Detection”](#) for more details.

11.2.1.7 DDRI SDRAM RCOMP Block

The DDRI RComp circuitry dynamically compensates for variations in operating conditions due to process, temperature, or voltage. These variations are measured through a resistive mechanism in a special I/O Pad and evaluated in the associated compensation circuitry. Adjustments are then made to the drive strength of the buffers ensuring error free operation over the entire range of operating conditions.

The DDRI RComp circuitry fine-tunes three separate I/O buffer parameters:

- P-Channel drive strength
- N-Channel drive strength
- Buffer slew rate

11.2.2 DDRI SDRAM Memory Support

The memory controller for the IXP45X/IXP46X network processors supports one or two banks of DDRI SDRAM. DDRI SDRAM allows zero data-to-data wait-state operation. DDRI SDRAM offers an extremely wide range of configuration options emerging from the SDRAMs internal interleaving and bursting capabilities.



The MCU supports only a 32-bit data bus width memory implementation (with and without ECC). The data bus width is controlled by the DDRI SDRAM Control Register.

The MCU supports DDRI SDRAM burst length of four. A burst length of four enables seamless read/write bursting of long data streams as long as the memory transaction does not cross the page boundary. Page boundaries are at naturally aligned boundaries. The MCU ensures that the page boundary is not crossed within a single transaction by initiating a disconnect at next ADB (128-byte address boundary) on the internal bus prior to the page boundary.

11.2.2.1 DDRI SDRAM Interface

The DDRI SDRAM interface signals generated by the memory controller unit are for DDRI SDRAM operation, there are no signals supporting DDRII SDRAM. Refer to the *Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet* for specifics on DDRI SDRAM interface signals.

The MCU SDRAM interface provides a flexible mix of combinations including:

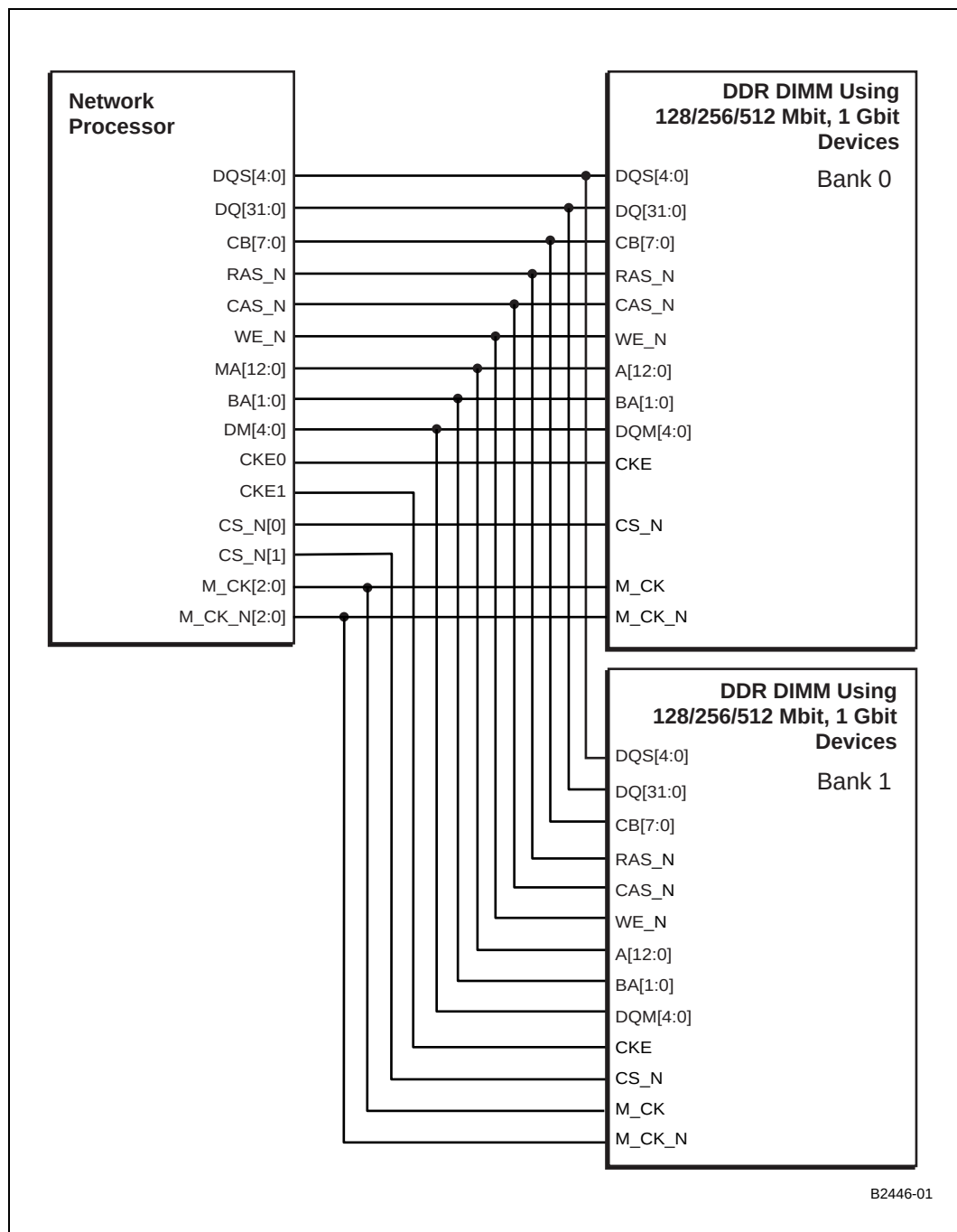
Table 202. DDRI SDRAM Memory Configuration Options

Data Bus Width	ECC Enabled	Maximum Throughput(1)
32 bit	Yes	1066 Mbyte/s
32 bit	No	1066 Mbyte/s
Note: 1. Based on DDRI 266 MHz SDRAM		

Figure 102 illustrates how two banks of DDRI SDRAM would interface with the IXP45X/ IXP46X network processors through the MCU.



Figure 102. Dual-Bank DDRI SDRAM Memory Subsystem





11.2.2.2 DDRI SDRAM Bank Sizes and Configurations

The MCU supports a memory subsystem ranging from 32 Mbyte to 1 Gbyte for 32-bit memory systems. An ECC or non-ECC system may be implemented using x8, or x16 devices. Table 203 illustrates the supported DDRI SDRAM configurations.

Table 203. Supported DDRI SDRAM Configurations

DDRI SDRAM Technology	DDRI SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size ¹	Page Size ²
			Row	Column	DDRI_BA[1]	DDRI_BA[0]		
128 Mbit ³	16 M x 8	1	12	10	ADDR[26]	ADDR[25]	64 M	4K
		2					128 M	4K
	8 M x 16	1	12	9	ADDR[25]	ADDR[24]	32 M	2K
		2					64 M	2K
256 Mbit	32 M x 8	1	13	10	ADDR[27]	ADDR[26]	128 M	4K
		2					256 M	4K
	16 M x 16	1	13	9	ADDR[26]	ADDR[25]	64 M	2K
		2					128 M	2K
512 Mbit	64 M x 8	1	13	11	ADDR[28]	ADDR[27]	256 M	8K
		2					512 M	8K
	32 M x 16	1	13	10	ADDR[27]	ADDR[26]	128 M	4K
		2					256 M	4K
1 Gbit ⁴	128 M x 8	1	14	11	ADDR[29]	ADDR[28]	512 M	8K
		2					1 G	8K
	64 M x 16	1	14	10	ADDR[28]	ADDR[27]	256 M	4K
		2					512 M	4K

1. Table indicates 32-bit wide memory subsystem sizes.
2. Table indicates 32-bit wide memory page sizes.
3. Supported with DDRI SDRAM only
4. Supported with DDRI SDRAM only

The 128/256/512-Mbit, 1-Gbit DDRI SDRAM devices comprise four internal leaves. The MCU controls the leaf selects within 128/256/512 Mbit, 1 Gbit DDRI SDRAM by toggling DDRI_BA[0] and DDRI_BA[1].

The two DDRI SDRAM chip enables DDRI_CS_N[1:0] support an DDRI SDRAM memory subsystem consisting of two banks. The base address for the two contiguous banks are programmed in the DDRI SDRAM Base Register (SDBR) and must be aligned to a 32 Mbyte boundary. The size of each DDRI SDRAM bank is programmed with the DDRI SDRAM boundary registers (SBR0 and SBR1). Bank 0 is also configurable into multiple regions.

Note: Systems must implement memory devices of all the same bus width (x8, x16) due to the single CAS address decode generated for a DDRI transaction. For example, systems implementing ECC and using x16 devices for the 32 bit DDRI data bus must also use a x16 device for the 8 bit ECC bus, even though 8 bits go unused.

Note: For IXP45X/IXP46X network processors, the DDRI SDRAM 32-Bit Size Register (S32SR) will not be used, since it will always be in 32-bit mode.

**Table 204. DDRI SDRAM Address Register Summary**

DDRI SDRAM Address Register	Definition
DDRI SDRAM Base Register (SDBR)	The lowest address for DDRI SDRAM memory space aligned to a 32-Mbyte boundary.
DDRI SDRAM Boundary Register 0 (SBR0)	The upper address boundary for bank 0 of DDRI SDRAM memory space. SBR0 must be greater than or equal to the value of SDBR[30:25].
DDRI SDRAM Boundary Register 1 (SBR1)	The upper address boundary for bank 1 of DDRI SDRAM memory space. SBR1 must be greater than or equal to SBR0.
DDRI SDRAM 32-Bit Size Register (S32SR)	The size for the memory space to operate as 32-bit memory (in MBs). S32SR must be less than, or equal to 1/2 of bank 0 size. Ignored with a 32-bit data bus width.

Note: DDRI SDRAM memory space must be aligned to a 32-Mbyte boundary and must *never* cross a 2-Gbyte boundary.

Note: With 32-bit DDRI SDRAM attached to the IXP45X/IXP46X network processors, all DDRI SDRAM memory space behaves as 32-bit DDRI SDRAM and the value in S32SR is ignored.

The base register defines the upper seven address bits of the DDRI SDRAM memory space. The boundary registers define the address limits for each DDRI SDRAM bank in 32 Mbyte granularity. [Table 205](#) defines the conditions which must be satisfied to activate a DDRI SDRAM memory bank.

Table 205. Address Decoding for DDRI SDRAM Memory Banks

Condition	DDRI SDRAM Bank Selected
ADDR[31] is not equal to the SDBR[31]	None
ADDR[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the SDBR[30:25] AD[30:25] is less than the value in SBR0	Bank 0
ADDR[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the value in SBR0 AD[30:25] is less than the value in SBR1	Bank 1
ADDR[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the value in SBR1	None

[Table 206](#) shows the correct programming values for the DDRI SDRAM Bank Size

Table 206. Programming Codes for the DDRI SDRAM Bank Size

Bank Size	Code	Bank Size	Code
Empty	00H	256 Mbyte	08H
32 Mbyte	01H	512 Mbyte	10H
64 Mbyte	02H	1 Gbyte	20H
128 Mbyte	04H		

encoding. These Bank Size Codes are used in the calculation of the DDRI SDRAM boundary registers programming values. [Equation 1](#) and [Equation 2](#) show the required equations to calculate the programming values for the DDRI SDRAM boundary registers.

Note: For IXP45X/IXP46X network processors, both banks must be programmed to be the same size.

**Equation 1. Programming Value for DDR1 SDRAM Boundary Register 0 (SBR0[6:0])**

$$\text{SBR0}[6:0] = \text{Bank 0 Size Code} + \text{SDBR}[30:25]$$

Equation 2. Programming Value for DDR1 SDRAM Boundary Register1 (SBR1[6:0])

$$\text{SBR1}[6:0] = \text{Bank 1 Size Code} + \text{SBR0}[6:0]$$

Table 207 shows the correct programming values for the 32-bit DDR1 SDRAM Size

Table 207. Programming Values for the DDR1 SDRAM 32-bit Size Register (S32SR[29:20]) *

32-bit Region Size	S32SR[29:20]	32-bit Region Size	S32SR[29:20]
Empty	000H	32 M	020H
1 M	001H	64 M	040H
2 M	002H	128 M	080H
4 M	004H	256 M	100H
8 M	008H	512 M	200H
16 M	010H	(reserved)	all other values
Note: For the IXP45X/IXP46X network processors, this is always programmed to 0.			

Register.

Example 21. Address Register Programming Example (Default Mode for IXP45X / IXP46X network processors)

The user wants to program the DDR1 SDRAM memory space to begin at 0000 0000H. Bank 0 is 128 Mbyte and Bank 1 is 128 Mbyte yielding a total memory of 256 Mbytes. All the memory is programmed for 32-bit mode, so there is no special 32-bit region using S32SR register. The memory space summary is:

Memory Space Limit	Address
DDR1 SDRAM Base	0000 0000H
32-Bit Region Top (end 32-bit region)	N/A
Invalid Region Top (start 64-bit region)	N/A
Bank 0 Top	07FF FFFFH
Bank 1 Top	0FFF FFFFH

The registers would be programmed as follows:

- Bank 0 Size = 128MB, code = 000100₂
- Bank 1 Size = 128MB, code = 000100₂
- SDBR = 0000 0000H, SDBR[30:25] = 000000₂
- SBR0[6:0] = 0000100₂ = 08H (size of Bank 0)
- SBR1[6:0] = 0001000₂ = 10H (size of Bank 0 + size of Bank 1)
- S32SR[29:20] = 000H, S32SR = 0000 0000H (same as default value)

Additionally, the following registers also need to be programmed before using the DDR1

- [DDR1 SDRAM Control Register 0 SDCR0](#) - Program according JEDEC specs.
- [DDR1 SDRAM Control Register 1 SDCR1](#) - Program according JEDEC specs.
- [ECC Control Register ECCR](#) - Program only if ECC is enabled



- Perform DDRI initialization sequence using [DDRI SDRAM Initialization Register SDIR](#) register.
- [Refresh Frequency Register RFR](#) - Program per JEDEC Spec using MCU clock of 133MHz

Note: All other registers can use their default register values for operation.

11.2.2.3 MTCTR Register Setup

In order to insure equal bandwidth between the MPI port and the two AHB ports, the MPTCR register must be programmed to 0x00000011H, other wise the default value will accept 12 MPI transactions to every one AHB transaction thereby starving the AHB ports of the memory controller.

11.2.2.4 DDRI SDRAM Addressing

[Table 208](#), [Table 209](#), and [Table 210](#) illustrate how the internal address is mapped to the DDRI_MA[13:0] lines for 128/256/512 Mbit, 1-Gbit DDRI SDRAM devices.

Table 208. DDRI SDRAM Address Translation for 128/512 Mbit (x16/x8), 1 Gbitx8, and 256 Mbitx8 Devices

DDRI_MA [13:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	ADDR[27]	I_AD[25]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	V ¹	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:

1. A10 is used for precharge variations on the read or write command. See [Table 211](#) for more details.
2. For the Leaf Selects, see [Table 203](#).

Table 209. DDRI SDRAM Address Translation for 256 Mbitx16 Devices

DDRI_MA [13:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	ADDR[27]	I_AD[24]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	V ¹	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:

1. A10 is used for precharge variations on the read or write command. See [Table 211](#) for more details.
2. For the Leaf Selects, see [Table 203](#).
3. 256 Mbitx16 addressing requires that **ADDR[24]** be presented as bit 12 of the row address instead of **ADDR[25]** as in [Table 208](#).



Table 210. DDRI SDRAM Address Translation for 1 Gbitx16 Devices

DDRI_MA [13:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	I_AD[26]	I_AD[25]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	V ¹	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:

1. A10 is used for precharge variations on the read or write command. See [Table 211](#) for more details.
2. For the Leaf Selects, see [Table 203](#).
3. 1 Gbitx16 addressing requires that **ADDR[26]** be presented as bit 13 of the row address instead of **ADDR[27]** as in [Table 208](#).

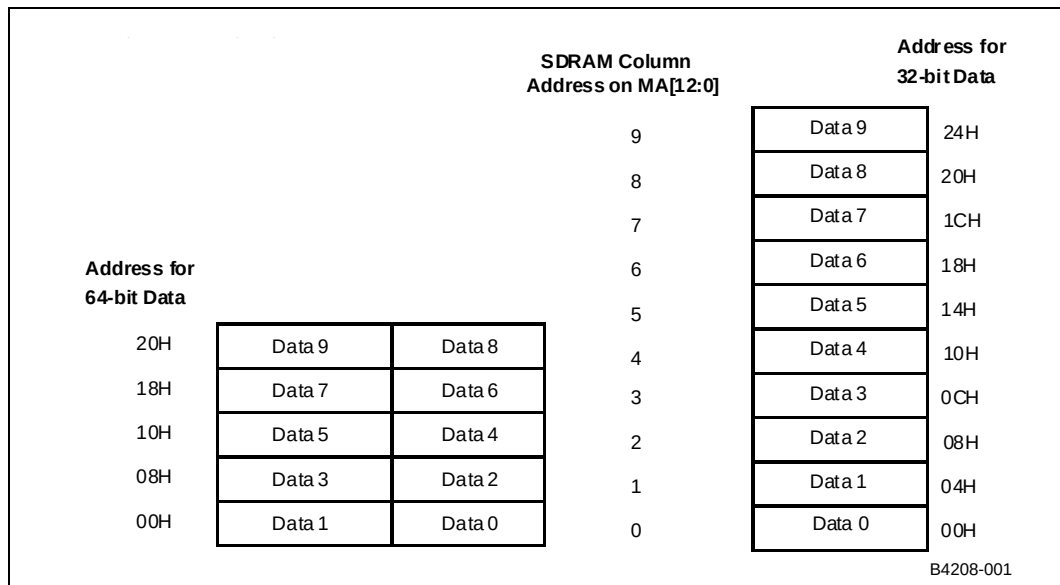
Since the MCU supports DDRI SDRAM bursting, the MCU increments the column address based on the burst length of four for each DDRI SDRAM read or write burst. The MCU supports a sequential and random burst types. Sequential bursting means that the address issued to the DDRI SDRAM is incremented by the DDRI SDRAM device in a linear fashion during the burst cycle. Random bursting means that the address issued to the DDRI SDRAM is any address in a currently active page.

11.2.2.5 32-Bit Data Bus Width

The MCU supports a 32-bit data bus width and a minimum memory size of 32 Mbytes, the maximum bus throughput is 1066 Mbytes/sec.

DDRI SDRAM address translation differs between 64- and 32-bit data bus widths. To generate a 32-bit address to the memory subsystem, the internal address ADDR[31:2] is shifted to the left by one bit prior to the address translations illustrated in [Table 208 through Table 210](#). This provides the granularity required for a 32-bit wide memory. See [Figure 103](#) for an example of how shifting the address before generating the DDRI SDRAM address on DDRI_MA[13:0] results in 32-bit addressing.

Figure 103. 64-Bit to 32-Bit Addressing





For example, assume an internal bus master issues a write to DDRI SDRAM memory space via the Internal Bus Memory Port. The address specified on ADDR[31:0] is 0000 0010H.

If 32-bit data bus width is enabled, the MCU shifts ADDR[31:2] by one bit to the left before translating the address to DDRI_MA[13:0]. Therefore, the column address becomes four for the first write transaction.

The MCU for the IXP45X/IXP46X network processors only supports a 32-bit data bus, although future products may support a 64-bit data bus width. The data bus width is selected by bit 2 of the SDCR (see “[DDRI SDRAM Control Register 0 SDCR0](#)” on [page 633](#)). The power-on default is a 64-bit bus width, but for the IXP45X/IXP46X network processors, the bus width **MUST** always be programmed to be 32-bit bus width.

11.2.2.6 Page Hit / Miss Determination

The page size is based on the DDRI technology as specified in [Table 203](#). The MCU keeps up to eight pages open simultaneously, one page each of:

- Bank0/Leaf0
- Bank0/Leaf1
- Bank0/Leaf2
- Bank0/Leaf3
- Bank1/Leaf0
- Bank1/Leaf1
- Bank1/Leaf2
- Bank1/Leaf3

[Figure 104](#) illustrates the logical flow of page hit determination

The MCU logic determines the hit/miss status for reads and writes. For a new DDRI SDRAM transaction, the MCU compares the address of the current transaction with the address stored in the appropriate page address register. Given the supported DDRI SDRAM devices and two banks, there are eight pages kept open simultaneously. The DDRI SDRAM chip enables DDRI_CS_N[1:0] and leaf selects DDRI_BA[1:0] determine which page address to compare.

If the current transaction misses the open page selected then the MCU closes the open page pointed to by DDRI_CS_N and DDRI_BA by issuing a **precharge** command. The MCU opens the current page with a **row-activate** command and the transaction completes with a **read** or **write** command. When the MCU opens the current page, the row address from the corresponding memory transaction queue is stored in the page address register pointed to by DDRI_CS_N and DDRI_BA so it may be compared for future transactions. See [Table 208](#), [Table 209](#), and [Table 210](#) for address mapping to row address.

Once the MARB issues a command to the MCU DDRI Control Block, the paging logic makes a hit/miss comparison. The performance is best for page hits and therefore the MCUs behavior is different for the hit and miss scenario.

For a page hit, the MCU does not need to open the page and avoids the RAS-to-CAS delay achieving greater performance. The waveform for a write including the row activation in the case of a page miss is illustrated in [Figure 115](#). For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.



If the current transaction hits the open page, then the page is already active and the **read** or **write** command may be issued without a **row-activate** command. When the next transaction is the same command type as the current transaction, and also a page hit, the MCU does not need to issue the command again, but simply drive column address for an open page.

If the refresh timer expires and the MCU issues an **auto-refresh** command, all pages are closed.

Figure 114 illustrates the performance benefit of a read hit versus a read miss in Figure 117. Figure 115 illustrates the performance benefit of a write hit versus a write miss in Figure 120.

Figure 104. Page Hit / Miss Logic for 128 / 256 / 512 / 1, 024-Bit Mode

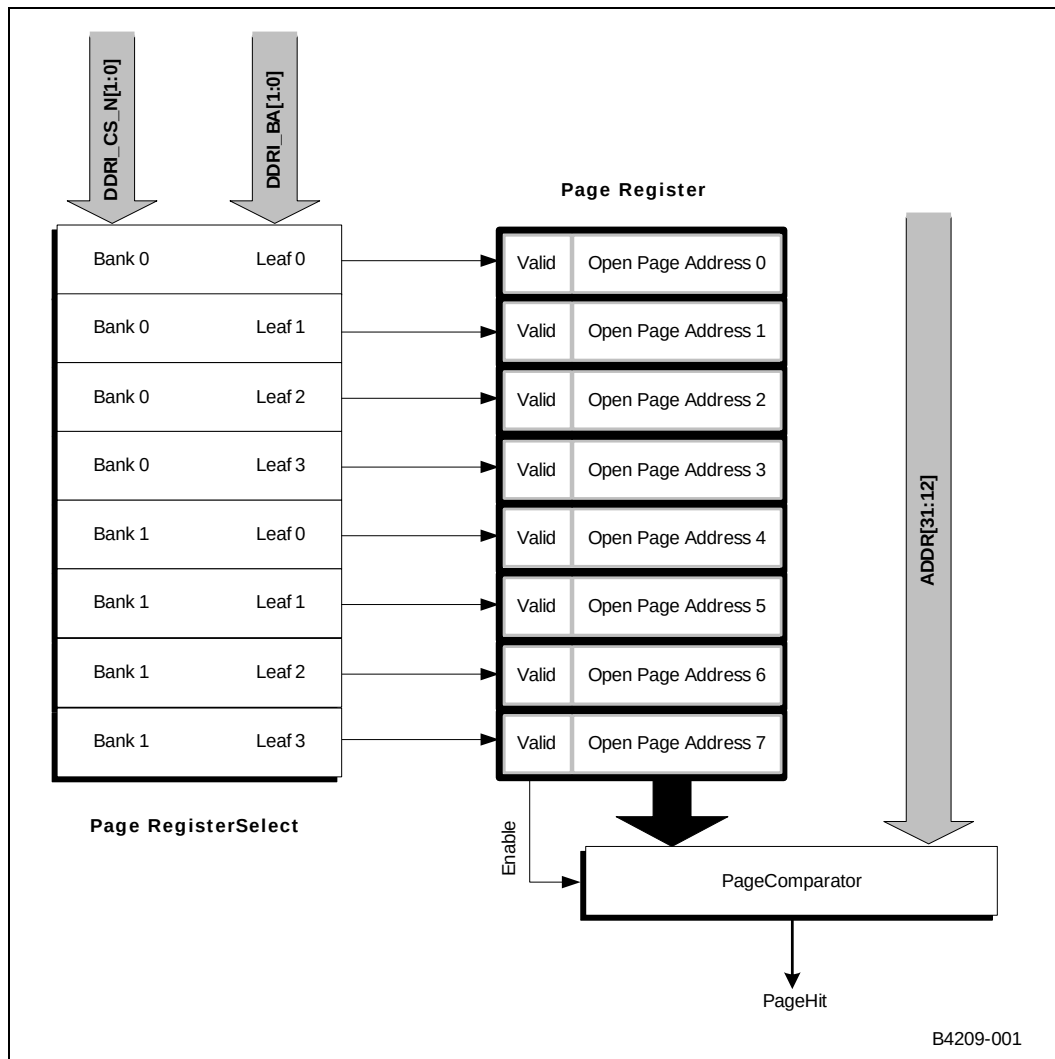




Figure 105. Logical Memory Image of a DDRI SDRAM Memory Subsystem

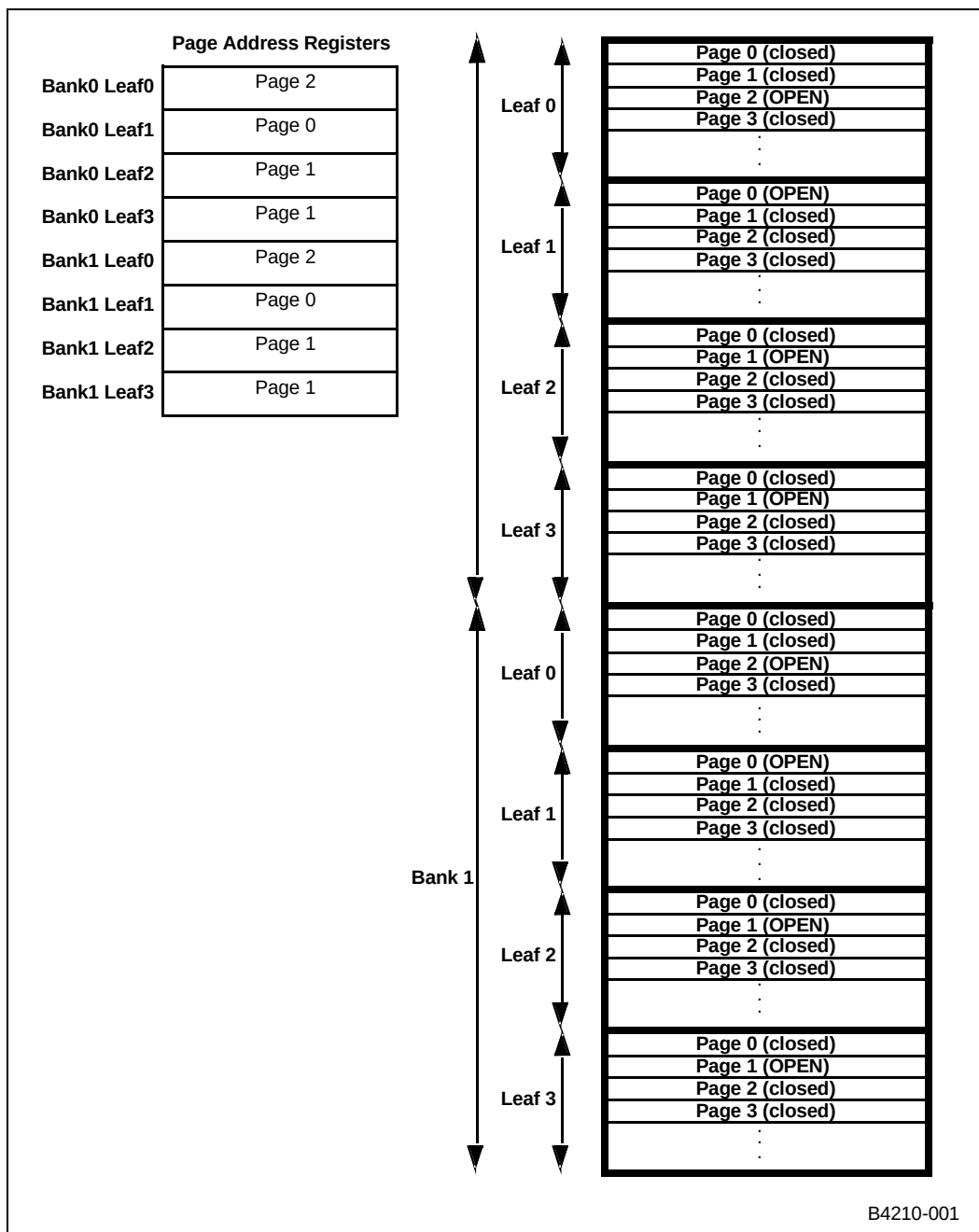


Figure 105 illustrates how the logical memory image is partitioned with respect to open and closed pages. If the above image represents a 128-Mbyte DDRI SDRAM memory size, each bank is 64 Mbytes and each leaf is 16 Mbytes.

Only one page may be open within each of the leaf blocks. The page sizes depend on the memory size implemented in the DDRI SDRAM memory subsystem as listed in Table 203. The programmer can optimize DDRI SDRAM transactions by partitioning code and data across the leaf boundaries to maximize the number of page hits.



11.2.2.7 DDRI SDRAM Commands

The MCU issues specific commands to the DDRI SDRAM devices by encoding them on the `DDRI_CS_N[1:0]`, `DDRI_RAS_N`, `DDRI_CAS_N`, and `DDRI_WE_N` inputs. [Table 211](#) lists all of the DDRI SDRAM commands understood by DDRI SDRAM devices. The MCU supports a subset of these commands.

Table 211. DDRI SDRAM Commands

Command ^{1,2}	Conditions					Comments
	<code>DDRI_CS_N[1:0]</code>	<code>DDRI_RAS_N</code>	<code>DDRI_CAS_N</code>	<code>DDRI_WE_N</code>	Other	
NOP	0	1	1	1		No Operation
Mode Register Set	0	0	0	0	<code>DDRI_BA[0] = Sel³</code> <code>DDRI_BA[1] = 0</code>	Load the (Extended) Mode Register from <code>DDRI_MA[13:0]</code>
Row Activate	0	0	1	1	<code>DDRI_BA[1:0] = Leaf</code>	Activate a row specified on <code>DDRI_MA[13:0]</code>
Read	0	1	0	1	<code>DDRI_BA[1:0] = Leaf</code> <code>DDRI_MA[10] = 0</code>	Column burst read Column address on <code>DDRI_MA[13:0]</code>
Read w/ Auto-Precharge	0	1	0	1	<code>DDRI_BA[1:0] = Leaf</code> <code>DDRI_MA[10] = 1</code>	Column burst read with row precharge at the end of the transfer
Write	0	1	0	0	<code>DDRI_BA[1:0] = Leaf</code> <code>DDRI_MA[10] = 0</code>	Column burst write Column address on <code>DDRI_MA[13:0]</code>
Write w/ Auto-Precharge	0	1	0	0	<code>DDRI_BA[1:0] = Leaf</code> <code>DDRI_MA[10] = 1</code>	Column burst write with row precharge at the end of the transfer
Precharge	0	0	1	0	<code>DDRI_BA[1:0] = Leaf</code> <code>DDRI_MA[10] = 0</code>	Precharge a single leaf
Precharge All	0	0	1	0	<code>DDRI_MA[10] = 1</code>	Precharge all leaves
Auto-Refresh	0	0	0	1		Refresh both banks from on-chip refresh counter
Self-Refresh	0	0	0	1	<code>CKE = 0</code>	Refresh autonomously while <code>CKE = 0</code>
Power Down	X	X	X	X	<code>CKE = 0</code>	Power down if both banks precharged when <code>CKE = 0</code>
Stop	0	1	1	0		Interrupt a read or write burst.

1. This table copied from *New DRAM Technologies* by Steven Przybylski.

2. Shaded boxes indicate commands not supported by IXP45X/IXP46X network processors. They are included for completeness.

3. During a Mode Register Set command, `DDRI_BA[1:0] = 002` selects the Mode Register, `DDRI_BA[1:0] = 012` selects the Extended Mode Register, all others are served.

DDRI SDRAM commands are synchronous to the clock so the MCU sets up the above conditions prior to the `DDRI_CK[2:0]` rising edge.

11.2.2.8 DDRI SDRAM Initialization

Since DDRI SDRAM devices contain a controller within the device, the MCU must initialize them specifically. Upon the deassertion of `RESET_IN_N`, software initializes the DDRI SDRAM devices with the sequence described below and illustrated in [Figure 109](#):

1. The MCU applies the clock `DDRI_CK[2:0]` at power up along with system power (clock frequency unknown).
2. The MCU must stabilize `DDRI_CK[2:0]` within 100 μ s after power stabilizes.
3. The MCU holds the following control inputs inactive:
`DDRI_RAS_N`, `DDRI_CAS_N`, `DDRI_WE_N`, `DDRI_CS_N[1:0]`



The MCU then places all of the following data outputs and strobes in the High-Z state:

DDR1_DQS[4:0], DDR1_DQ[31:0], DDR_CB[7:0]

The MCU deasserts DDR1_CKE[1:0] for a minimum of 200 us after supply voltage reaches the desired level. Asserting PWRON_RESET_N achieves this state.

4. The MCU then performs sixteen RCOMP calibration cycles which take an additional 34 ms. Software MUST wait until this time has elapsed before continuing with DDR1 SDRAM initialization.
5. Software overwrites the default value of register DDR_RCOMP_CSR3 (Hex Offset Address = 0x0CC00 F570) with a new value of 0x0000 1000.
6. Software overwrites the default value of register DDR_DRIVE3 (Hex Offset Address = 0x0CC00 F5AC) with a new value of 0x0002 08F0.
7. Software disables the refresh counter by setting the RFR to zero.
8. Software issues one **NOP** cycle after the 200 us device deselect. A **NOP** is accomplished by setting the SDIR to 0011₂. The MCU asserts DDR1_CKE[1:0] with the NOP.
9. Software issues a **precharge-all** command to the DDR1 SDRAM interface by setting the SDIR to 0010₂.
10. Software issues an **extended-mode-register-set** command to enable the DLL by writing 0100₂ to the SDIR. The MCU supports the following DDR1 SDRAM mode parameters:
 - a. DLL = Enable/Disable
 - b. Additive Latency (AL) is always zero for the IXP45X/IXP46X network processors. The MCU only supports an AL of zero because the MCU does not support back-to-back Active to Read or Write commands that would otherwise be in violation of tRCD (Active to Read/Write command delay): tRCD is obeyed at all times. Thus, Write Latency (WL) is always defined as shown in [Figure 112 on page 605](#).

Figure 106. Supported DDR1 SDRAM Extended Mode Register Settings

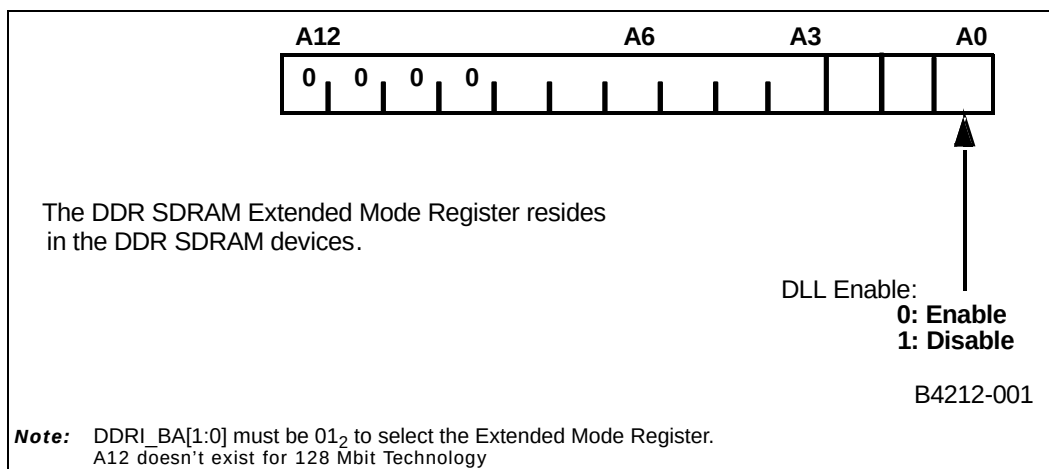
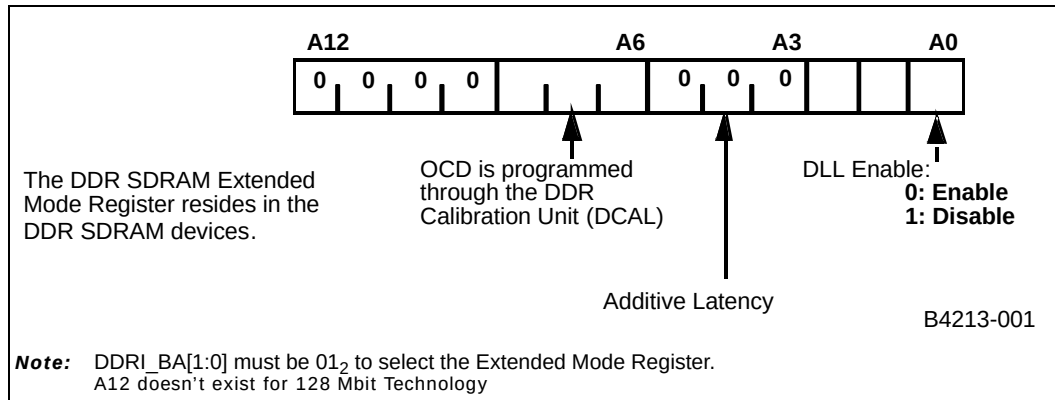
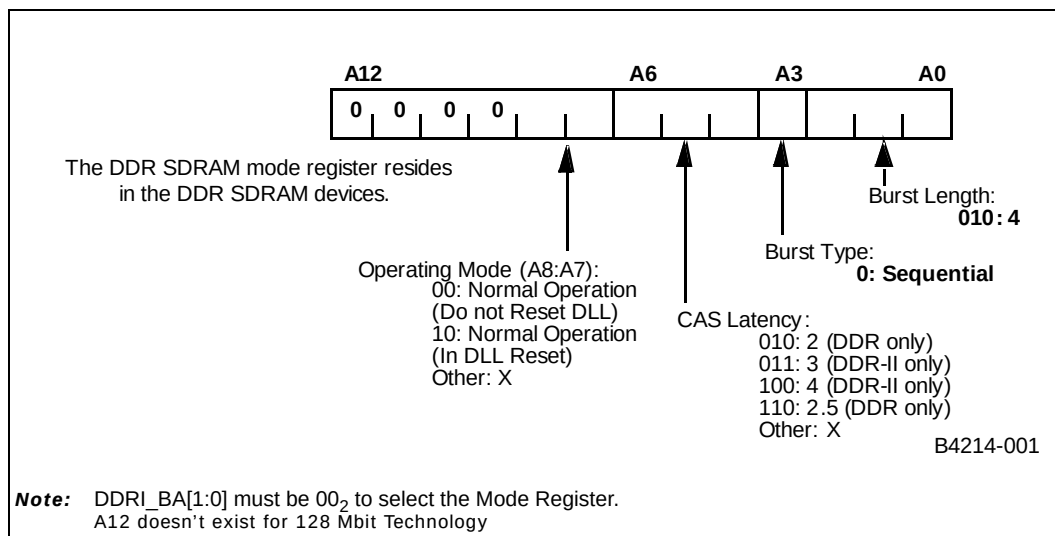


Figure 107. Supported DDRII SDRAM Extended Mode Register Settings



11. After waiting T_{mrd} cycles, software issues a **mode-register-set** command by writing 0001₂ to the SDIR to program the DDRI SDRAM parameters and to **reset** the DLL. The MCU supports the following DDRI SDRAM mode parameters:
 - a. CAS Latency (t_{CAS}) = two or two and one-half for DDRI, or three or four for DDRII based on the programmed setting in [Section 11.6.2, “DDRI SDRAM Control Register 0 SDCR0”](#).
 - b. Burst Type = Sequential
 - c. Burst Length (BL) = four

Figure 108. Supported DDRI SDRAM Mode Register Settings



12. After waiting T_{mrd} cycles, software issues a **precharge-all** command to the DDRI SDRAM interface by setting the SDIR to 0010₂.
13. After waiting T_{rp} cycles, software provides two **auto-refresh** cycles. An **auto-refresh** cycle is accomplished by setting the SDIR to 0110₂. Software must ensure at least T_{rfc} cycles between each **auto-refresh** command.
14. Following the second **auto-refresh** cycle, software must wait T_{rfc} cycles. Then, software issues a **mode-register-set** command by writing to the SDIR to program the DDRI SDRAM parameters **without** resetting the DLL by writing 0000₂ to the SDIR.

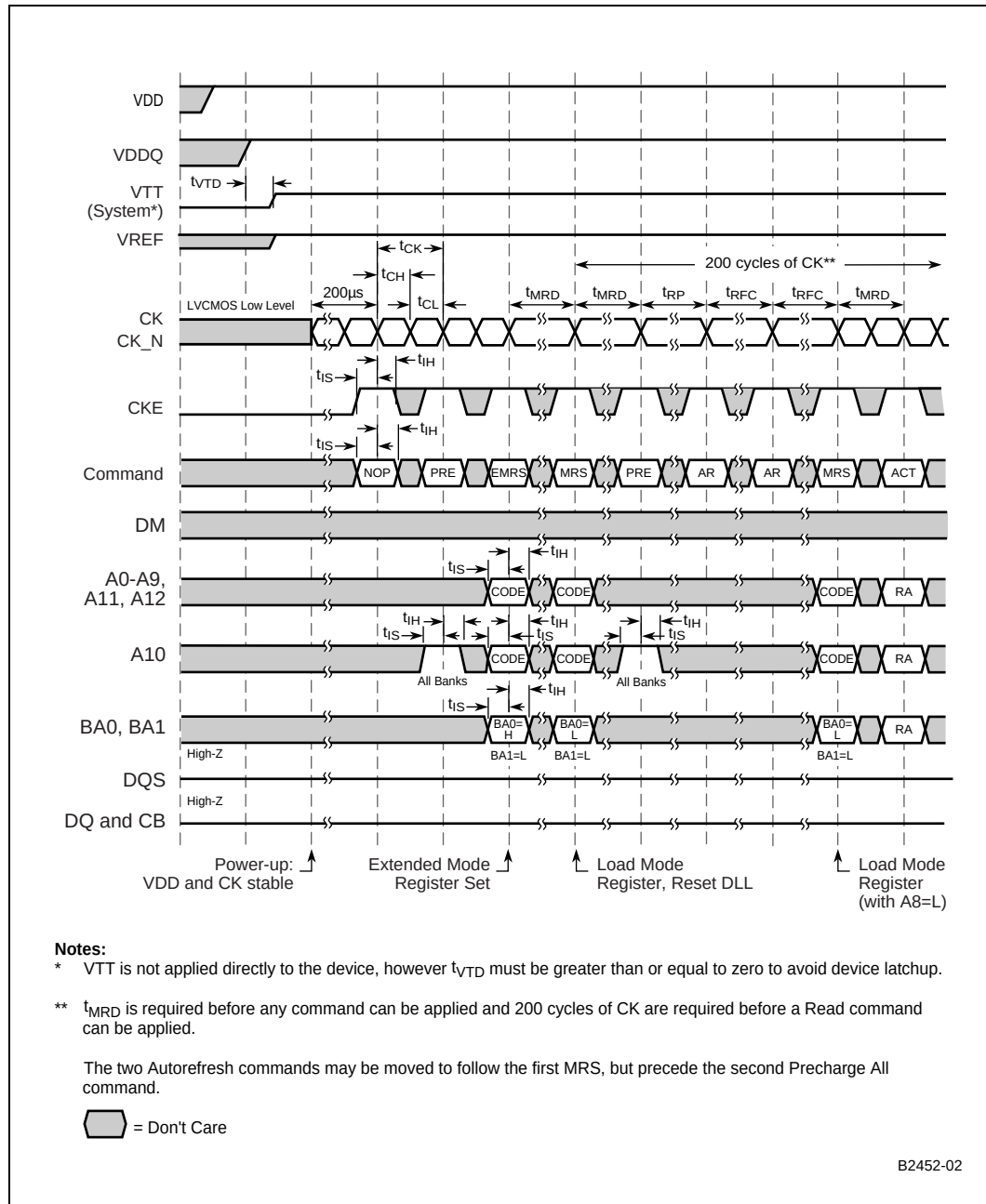


15. The MCU may issue a **row-activate** command T_{mrd} cycles after the **mode-register-set** command.

16. Software re-enables the refresh counter by setting the RFR to the required value.

The waveform in [Figure 109](#) illustrates the DDRI SDRAM initialization sequence.

Figure 109. DDRI SDRAM Initialization Sequence (Controlled with Software)



If the DDRI SDRAM subsystem implements ECC (see [Section 11.2.3, "Error Correction and Detection"](#)), then initialization software must initialize the entire memory array with the IXP45X/IXP46X network processors. It is important that every memory



location has a valid ECC byte. The Intel XScale processor must be used to fill the memory array with a constant, thereby initializing the associated ECC bytes in the process. If the memory array is not initialized, the BIU or PCC may attempt to read memory locations beyond the specified word(s). In this case, the MCU will report an ECC error even though software did not specifically request the un-initialized data.

11.2.2.9 DDRI SDRAM Mode Programming

The MCU programs the DDRI SDRAM devices through a **mode-register-set** command. During the initialization sequence this command sets the DDRI SDRAM mode register (see [Section 11.2.2.8, “DDRI SDRAM Initialization”](#)) by programming the SDIR and SDCR[1:0].

The DDRI SDRAM state machine ensures that a **row-activate** command is issued no sooner than T_{mrd} cycles after the **mode-register-set** command.

The values to be programmed in the SDCR[1:0] registers are based on the DDRI SDRAM devices being interfaced to the IXP45X/IXP46X network processors. Because the parameters that define the time between allowed commands are programmable, this allows flexibility in the type of DDRI device that is selected, in addition to decoupling the hardware to any frequency dependencies.

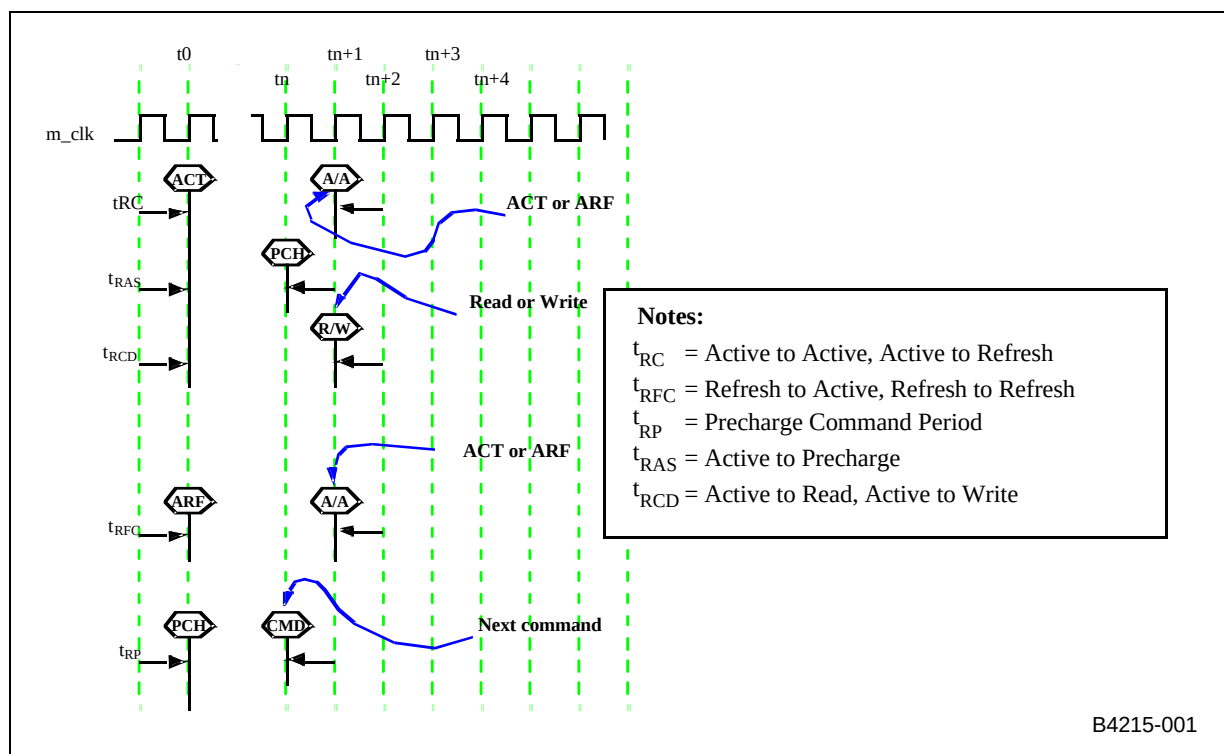
Note: The MCU_DDRSM will NOT interact properly with the DDRI SDRAM until the SDCR[1:0] registers have been programmed.

The duration between valid commands must be programmed by the user before the DDRSM can interact properly with the DDRI. These parameters are defined by either JEDEC and/or within this document in the timing diagrams and equations in [Section 11.6.2, “DDRI SDRAM Control Register 0 SDCR0”](#) and [Section 11.6.3, “DDRI SDRAM Control Register 1 SDCR1”](#).

The timing parameters for all non-read and non-write commands are derived directly from the *JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79*, June 2004 and *JEDEC DDR - II SDRAM Specification*, September 2002. These parameters, and their relationship to each other, are outlined in [Figure 110](#). Please see the device specification for the timing parameters specific to the DDR device that is to be implemented in the system.



Figure 110. MCU Active, Precharge, Refresh Command Timing Diagram



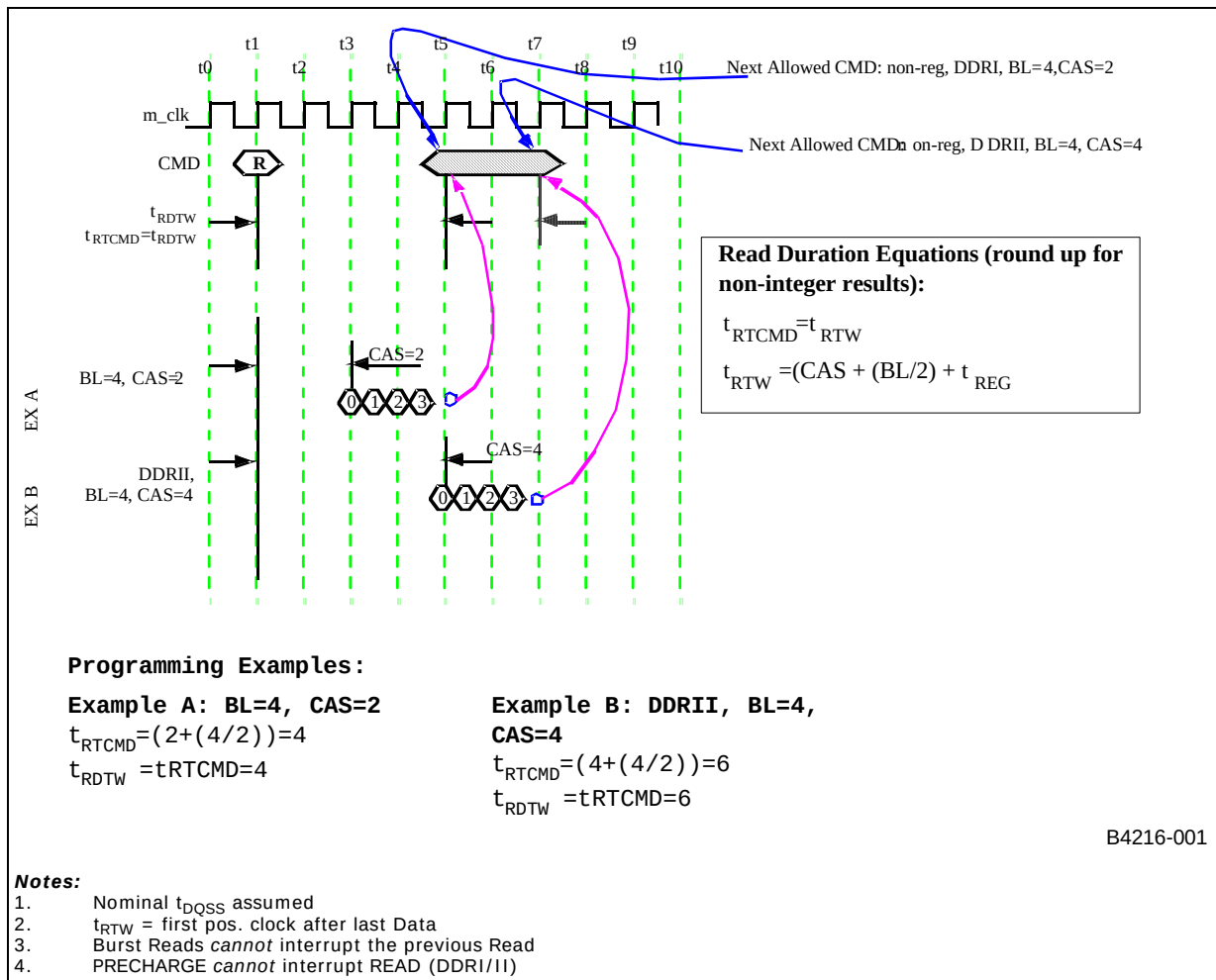
The timing parameters for DDR reads are defined in Figure 111. Both Read to Write (IXP45X/IXP46X network processors t_{RTW}) and Read to Command (IXP45X/IXP46X network processors t_{RTCMD}) are defined the same. It is important to note that they remain separate for two reasons: 1) For similarity to the DDR Write timing parameters, and 2) to allow for flexibility in programming the MCU that might not have been comprehended at the time of its design. Both parameters take into account CAS latency (**JEDEC**: t_{CAS}) and Burst Length (**JEDEC**: BL).

Note: Burst Length is fixed at four for the IXP45X/IXP46X network processors.

Note: The MCU allows for back-to-back reads, so long as they are to a currently open page.



Figure 111. MCU DDR Read Command to Next Command Timing Diagram



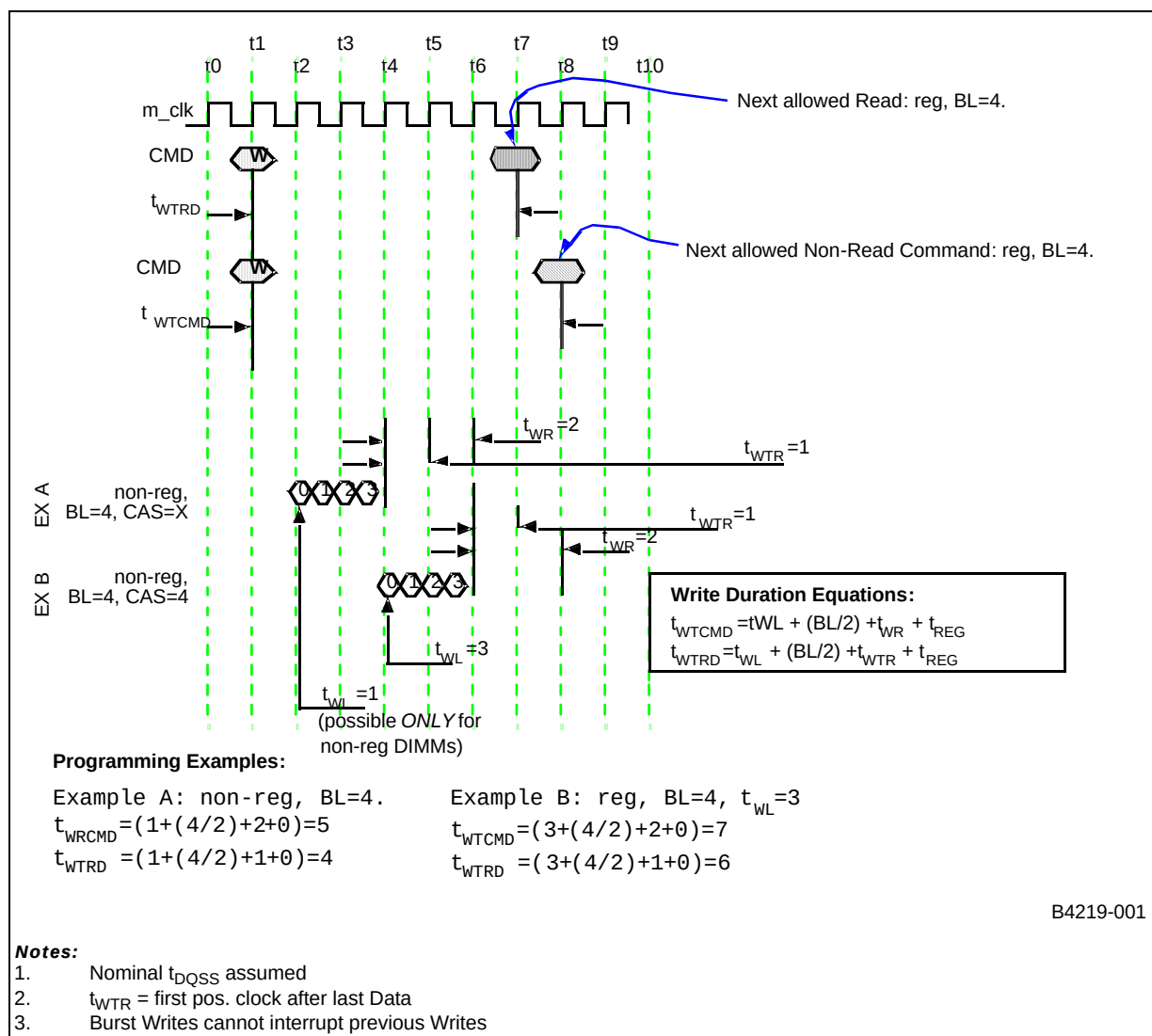
The timing parameters for DDR writes are defined in Figure 112. Both Write to Read (IXP45X/IXP46X network processors: t_{WTRD}) and Write to Command (IXP45X/IXP46X network processors: t_{WTCMD}) are defined in much the same manner, both accounting for Write Latency (JEDEC: t_{WL}) and Burst Length (JEDEC: BL). The difference lies in the compensation for Write Recovery (JEDEC: t_{WR}) and Write to Read (JEDEC: t_{WTR}).

Note: Burst Length is fixed at four for the IXP45X/IXP46X network processors.

Note: The MCU allows for back-to-back Writes, so long as they are to an open page.



Figure 112. MCU DDR Write Command to Next Command Timing Diagrams



B4219-001



11.2.2.10 DDRI SDRAM Read Cycle

The MCU performance is optimized for page hits and the MCUs behavior is different for the hit and miss scenario.

The waveform for a read including the row activation in the case of a page miss is illustrated in Figure 114. For a page hit, the two cycles required for row activation are saved resulting in lower first word read latency.

The MCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDRI SDRAM Control Block to issue the transaction command to the DDRI array if the previous transaction is the same type (read or write). In addition, the DDRI SDRAM Control Block supports pipelining of transactions which allows the column address of the next transaction to be issued before the current transaction's data transfer is completed by the DDRI SDRAM devices. These optimizations are illustrated in Figure 113 for random read memory transactions.

Figure 113. DDRI SDRAM Pipelined Reads

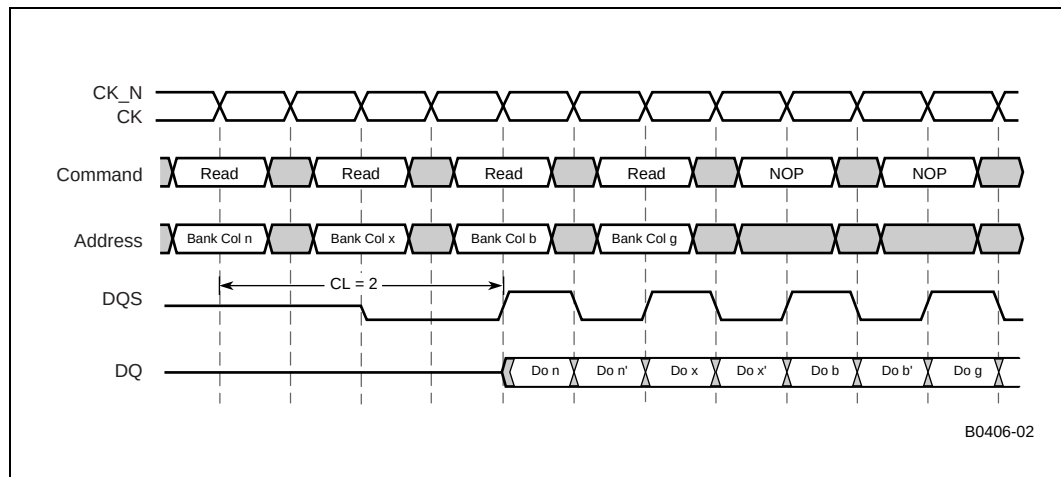
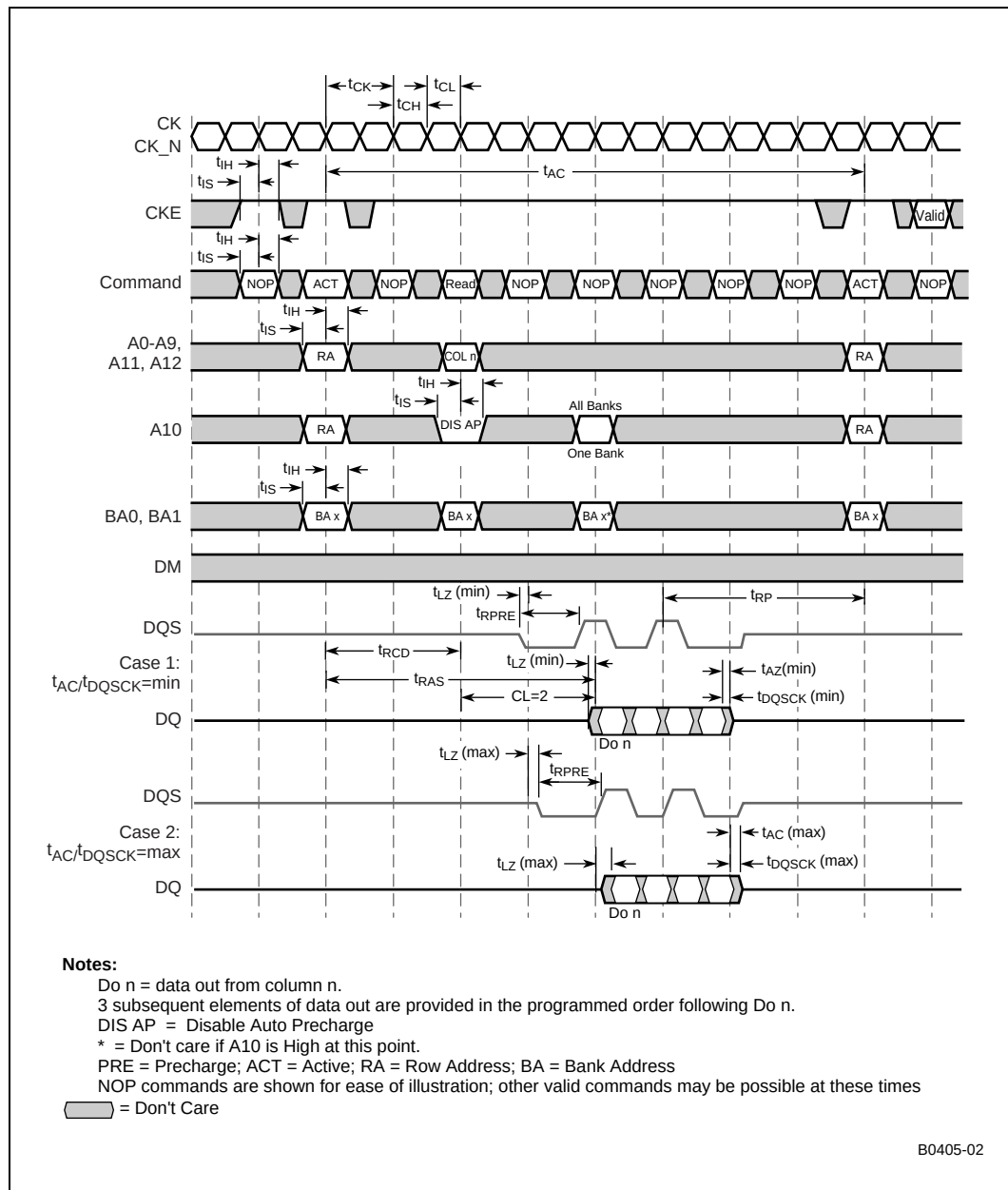




Figure 114. DDR1 SDRAM Read, 36 Bytes, ECC Enabled, BL=4



- Each of the MCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
 - If the address falls in the DDR1 SDRAM address range indicated by the SDBR, SBR0, SBR1, and S32SR, the MCU claims the transaction and latches the transaction in the respective memory transaction queue.
- Once the MARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDR1 SDRAM control block. The DDR1 SDRAM Control Block decodes the address to determine whether or not any of the open pages are hit.



A read that misses the open pages encounters a miss penalty because the currently open page needs to be closed before the read can be issued to the new page. Refer to [Section 11.2.2.6, “Page Hit/Miss Determination”](#) for the paging algorithm details. If a page hit occurs, steps 3-4 are skipped by the MCU.

3. The DDRI SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row. (Not depicted in [Figure 114](#).)
 - The DDRI SDRAM Control Block waits T_{rp} cycles after the precharge before issuing the **row-activate** command for the new read transaction.
4. The **row-activate** command enables the appropriate row.
 - The DDRI SDRAM Control Block asserts `DDRI_RAS_N`, de-asserts `DDRI_WE_N`, and drives the row address on `DDRI_MA[13:0]`.
5. In the following cycle in the case of a page hit or after T_{rcd} cycles in the case of a page miss, the DDRI SDRAM Control Block asserts `DDRI_CAS_N`, de-asserts `DDRI_WE_N`, and places the column address on `DDRI_MA[13:0]`. This initiates the burst read cycle.
6. After the CAS latency expires, the DDRI SDRAM device drives data to the MCU. A CAS latency of 2 is depicted in [Figure 114](#).
7. Upon receipt of the data, the DDRI SDRAM Control Block calculates the ECC code from the data and compares it with the ECC returned by the DDRI SDRAM array. [Section 11.2.3, “Error Correction and Detection”](#) explains the ECC algorithm in more detail.
8. Assuming the calculated ECC matches the read ECC, the DDRI SDRAM Control Block drives the data back to the corresponding memory transaction queue.
 - For each burst read issued, the memory controller increments the column address by four.

The MCU continues to return data to the corresponding memory transaction queue based on the byte count of the transaction.

Note: The burst cycle continues within the DDRI SDRAM devices unless the next read/write transaction is ready to be issued. The current DDRI SDRAM burst read may be interrupted by issuing the next **read** or **write** command.

The MCU divides transactions if they cross a page boundary. For read transactions, the transactions will be completed in multiple completions which end at page boundaries. Write transactions will receive posted data to the inbound memory transaction queue up to a full posted write queue (32 Bytes). Multiple write commands will then be issued to the DDRI SDRAM to complete the write of data to the multiple pages.



11.2.2.11 DDRI SDRAM Write Cycle

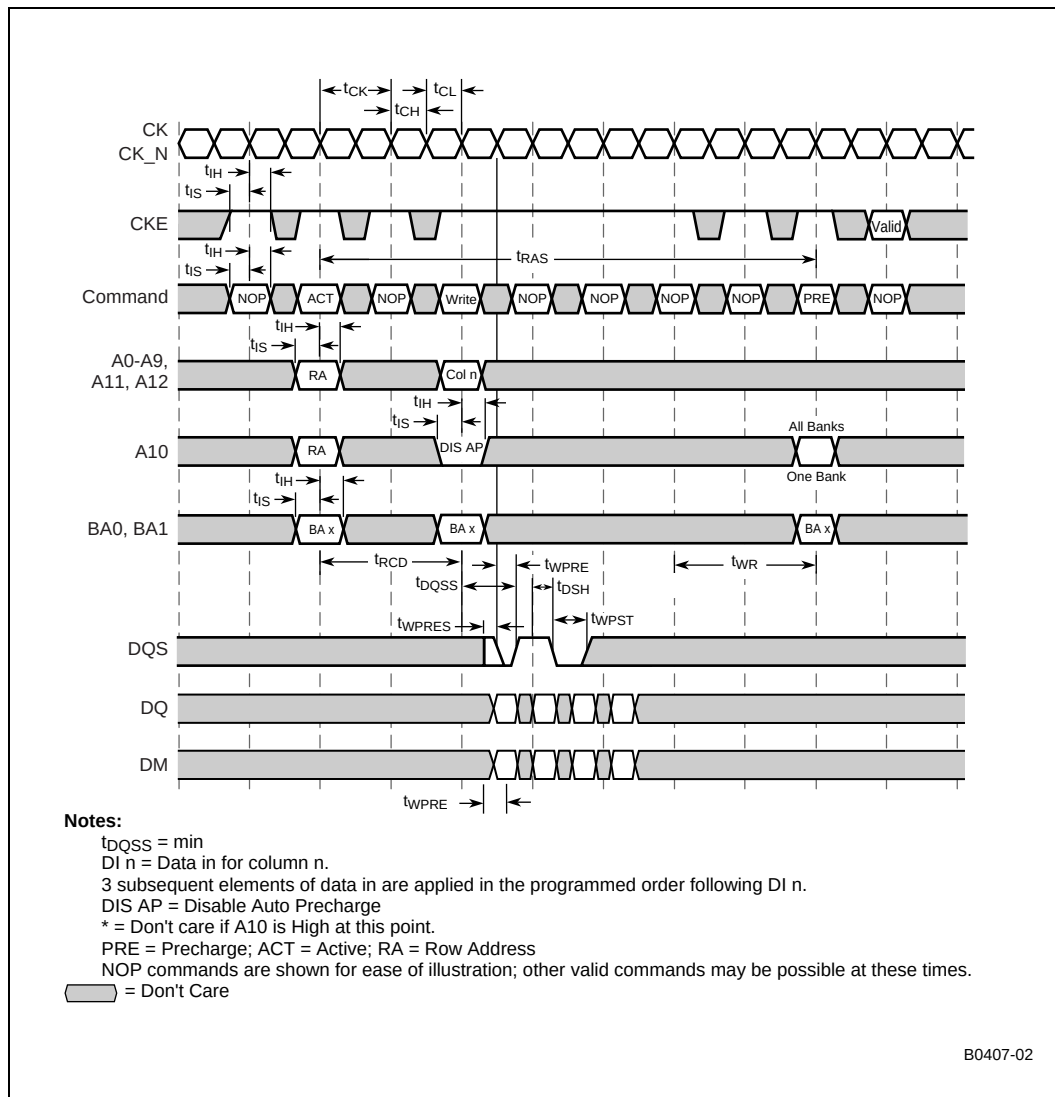
All write transactions to the DDRI SDRAM are posted to the MCU in the memory transaction queues. This implies that the transaction completes between a given port and corresponding unit prior to data being written to the SDRAM array. Once the MARB issues a **write** command from a memory transaction queue to the MCU DDRI Control Block, the paging logic makes a hit/miss comparison (illustrated in [Figure 104](#)). The performance is best for page hits and therefore the MCUs behavior is different for the hit and miss scenario.

Write transactions require ECC codes to be generated and stored in the SDRAM array with the data being written. The behavior is different depending on the size of the data being written. "[Error Correction and Detection](#)" on [page 614](#) explains the ECC algorithm in more detail.

For a page hit, the MCU does not need to open the page (assert DDRI_RAS_N) and avoids the RAS-to-CAS delay achieving greater performance. The waveform for a write including the row activation in the case of a page miss is illustrated in [Figure 115](#). For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.



Figure 115. DDR1 SDRAM Write, 36 Bytes, ECC Enabled, BL=4





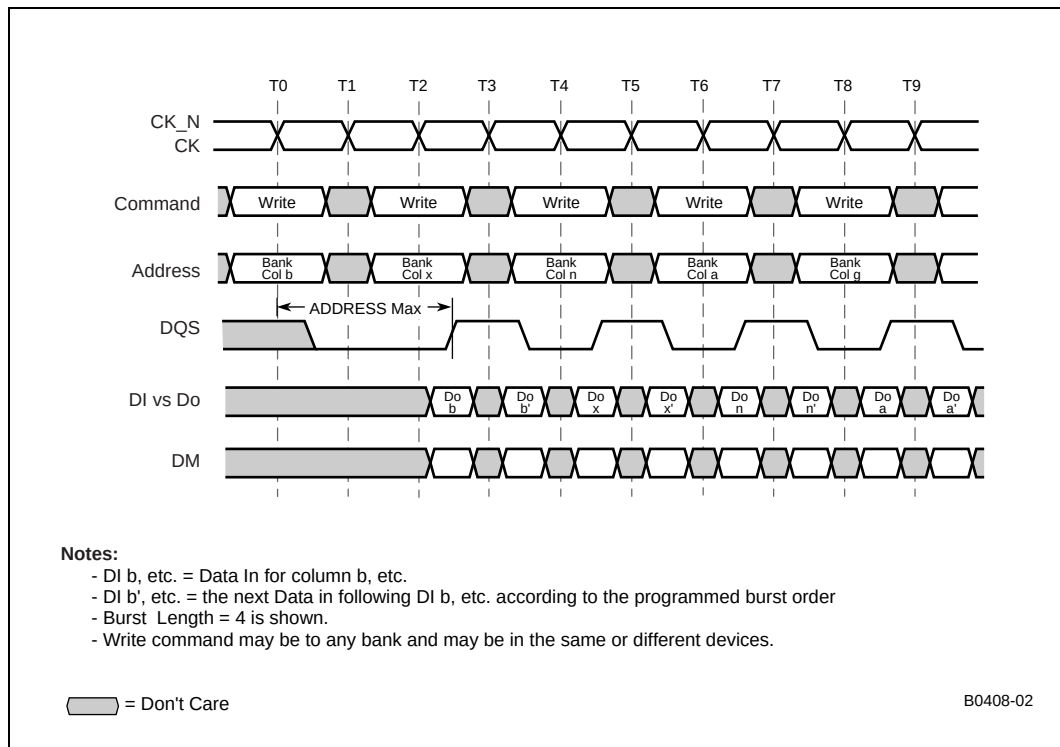
1. Each of the MCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
 - If the address falls in the DDRI SDRAM address range indicated by the SDBR, SBR0, SBR1, and S32SR the MCU claims the transaction and latches the transaction in the respective memory transaction queue.
2. Once the MARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDRI SDRAM control block. The DDRI SDRAM Control Block decodes the address to determine whether or not any of the open pages are hit.
 - The ECC logic generates the ECC code for the data to be written.

A write that misses the open page encounters a miss penalty because the currently open page needs to be closed before the write can be issued to the new page. Refer to [“Page Hit/Miss Determination” on page 595](#) for the paging algorithm details. If a page hit occurs, steps 3-4 are skipped by the MCU.

3. The DDRI SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row.
 - The DDRI SDRAM Control Block waits T_{rp} cycles after the precharge before issuing the **row-activate** command for the new write transaction.
4. The **row-activate** command enables the appropriate row.
 - The DDRI SDRAM Control Block asserts DDR_RAS_N , de-asserts DDR_WE_N , and drives the row address on $\text{DDR\_MA}[13:0]$.
5. After T_{rcd} cycles in the case of a page miss, the DDRI SDRAM Control Block asserts DDR_CAS_N , asserts DDR_WE_N , and places the column address on $\text{DDR\_MA}[13:0]$. This initiates the burst write cycle. The DDRI SDRAM Control Block drives the data to be written and its ECC code to the DDRI SDRAM devices.
 - The DDRI SDRAM Control Block drives the new data to the corresponding memory transaction queue each cycle until the transaction is completed with the byte count expiring, or the transaction is interrupted if preemption conditions are met.
 - For each burst issued, the DDRI SDRAM Control Block increments the address by four.
 - If ECC is enabled, when the data to write is not aligned on an 4 byte boundary for 32-bit data bus width, the DDRI SDRAM Control Block will perform a read-modify-write of the entire 4 byte aligned double-word for 32-bit data bus width and incorporate the new data while regenerating ECC.

Note: Even though the data is not written while the DDRI SDRAM Control Block asserts $\text{DDR\_DM}[4:0]$, the burst cycle will complete.

The MCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDRI SDRAM Control Block to issue the transaction command to the DDRI array if the previous transaction is the same type (read or write). In addition, the DDRI SDRAM Control Block supports pipelining of transactions which allows the column address of the next transaction to be issued before the current transaction's data transfer is completed by the DDRI SDRAM devices. These optimizations are illustrated in [Figure 116](#) for random write memory transactions.

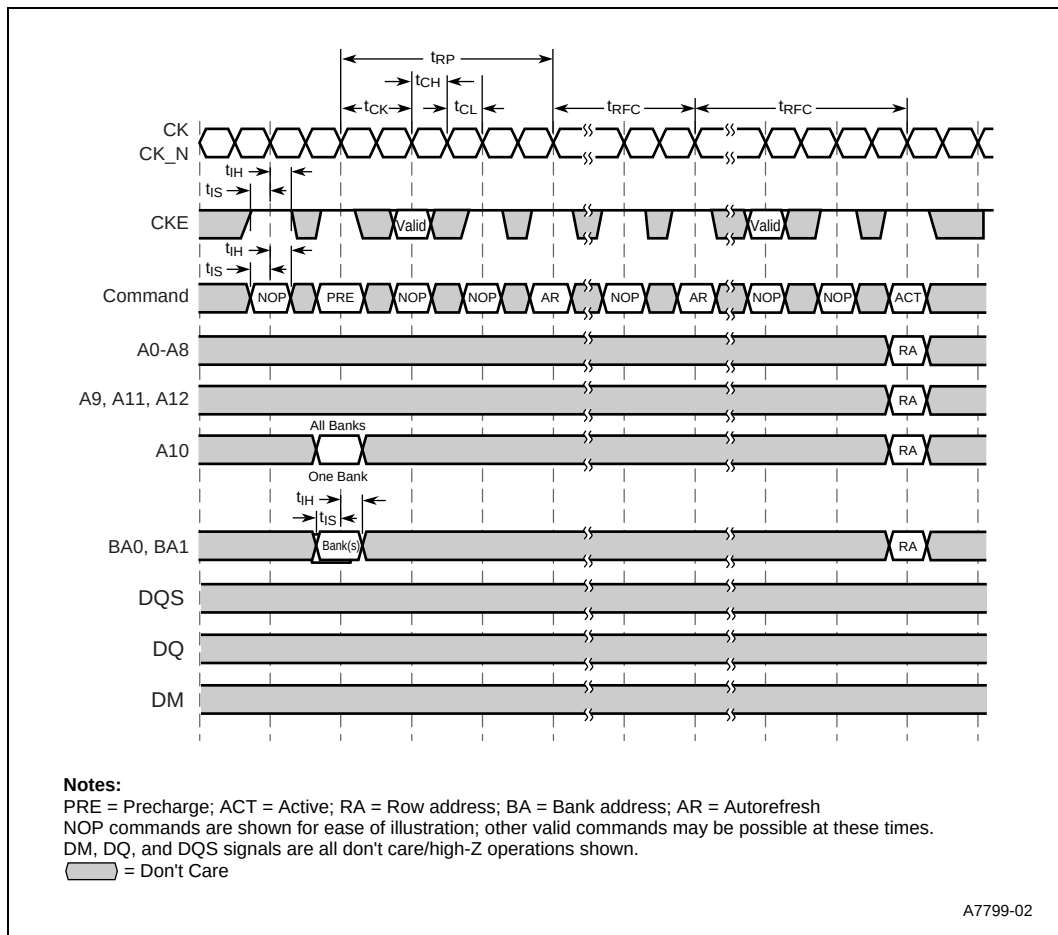
Figure 116. DDRI SDRAM Pipelined Writes


11.2.2.12 DDRI SDRAM Refresh Cycle

Since the DDRI SDRAM is a dynamic memory, the MCU issues a refresh cycle periodically. The interval of these refresh cycles is programmable in the RFR register. The DDRI SDRAM device generates the refresh address internally. The MCU initiates two sequential refresh cycles (one per bank) after the MCUs refresh timer expires and any current transaction is complete. To preserve the correct refresh period, the refresh timer continues counting after it expires to prevent a gradual skewing of the refresh interval. The waveform in [Figure 117](#) illustrates the case where the refresh timer expires while the memory bus is not busy.



Figure 117. Refresh While the Memory Bus is Not Busy



- Once the refresh timer expires, the MCU knows that a refresh cycle is necessary.
 - The refresh timer continues to count for the next refresh cycle.
- The MARB allows the current transaction to complete.
 - If the DDRI SDRAM Control Block and the DDRI SDRAM array are transferring data, or a CMTQ tenure is ongoing, the refresh cycle is queued until the transaction is complete or the CMTQ tenure expires.
- The DDRI SDRAM Control Block closes all open pages with a **precharge-all** command to all the populated DDRI SDRAM banks.
 - The DDRI SDRAM Control Block resets the page register valid bits.
- The DDRI SDRAM Control Block issues **auto-refresh** command to DDRI SDRAM bank 0.
 - This command affects all internal leaves.
- In the next cycle, the DDRI SDRAM Control Block issues **auto-refresh** command to DDRI SDRAM bank 1.
 - The MCUs internal 2-bit refresh counter is decremented by one. This is actually done when the DDRSM consumes the command from the ARB.



- After T_{rfc} cycles, the DDRI SDRAM Control Block can service a new transaction or another refresh cycle.

It is recommended that the RFR is programmed with the value to achieve 7.8 μ s, though some DDRI SDRAM devices may provide for the ability to refresh at a period of 15.6 μ s. See [Section 11.6.14, “Refresh Frequency Register RFR”](#) for details. The value is based on the frequency of the DDRI SDRAM and [Table 212](#) can provides for these two typical values.

Table 212. Typical Refresh Frequency Register Values

DDRI Speed	7.8 μ s Value	15.6 μ s Value
266 MHz	0410H	0820H

The longest possible internal bus transaction is writing a 32-byte burst where each data cycle results in a read-modify-write due to partial writes. See [Section 11.2.3.2, “ECC Generation for Partial Writes”](#) for details. The longest possible CMTQ tenure is 16 transactions where each of the transaction are page misses and partial writes.

The MCU uses a 2-bit counter to queue refresh cycles. If another refresh cycle is queued when a current refresh cycle completes, the MCU services the queued refresh cycle before servicing any internal bus requests. If the counter exceeds three (which should never happen), the counter does NOT roll over and three refreshes will be serviced when the MCU is available.

11.2.3 Error Correction and Detection

The MCU is capable of correcting any single bit errors and detecting any double bit errors in the's DDRI SDRAM memory subsystem for the IXP45X/IXP46X network processors. ECC enhances the reliability of a memory subsystem by correcting single bit errors caused by electrical noise or occasional alpha particle hits on the DDRI SDRAM devices.

Similar to parity, which simply detects single bit errors, error correction requires an additional 8-bit code word for the 32-bit datum. This means that a memory must have the additional 8-bit error correction code DDRI_CB[7:0] per 32-bit datum DDRI_DQ[31:0] resulting in a 40-bit wide memory subsystem. During DDRI SDRAM read cycles, the DDRI SDRAM Control Block detects single-bit errors and corrects the data prior to returning the data to the respective memory transaction queue. DDRI SDRAM write cycles generate the ECC and sends it with the data to the memories.

In 32-bit wide memory, the IXP45X/IXP46X network processors will zero extend the 32-bit datum to a 64-bit datum in order to generate, check and correct ECC. This means that a 32-bit datum memory with ECC will result in a 40-bit wide memory since an 8-bit error correction code is still required.

The ECC algorithm for the IXP45X/IXP46X network processors uses the H-matrix/G-Matrix method used by the Pentium® Pro Processor ([Figure 122](#)). The H-matrix reads a non-zero syndrome and indicates which bit was incorrect for single-bit errors.

Scrubbing is the process of correcting an error in the memory array. The chance of an unrecoverable multi-bit error increases if the software does not correct a single-bit error in the array. For the IXP45X/IXP46X network processors, scrubbing is handled by software. If error reporting is enabled, the MCU logs the error type in ELOG0 or ELOG1 and the address in ECAR0 or ECAR1 when an error occurs.



11.2.3.1 ECC Generation

For write operations, the MCU generates the error correction code which is written along with the data. This section describes the operation of the DDRI SDRAM Control Block for ECC generation in a 64-bit wide memory and 64-bit region. The same principles apply for 32-bit wide memory, though the MCU will generate 8-bit wide ECC by zero extending the data to 64-bits.

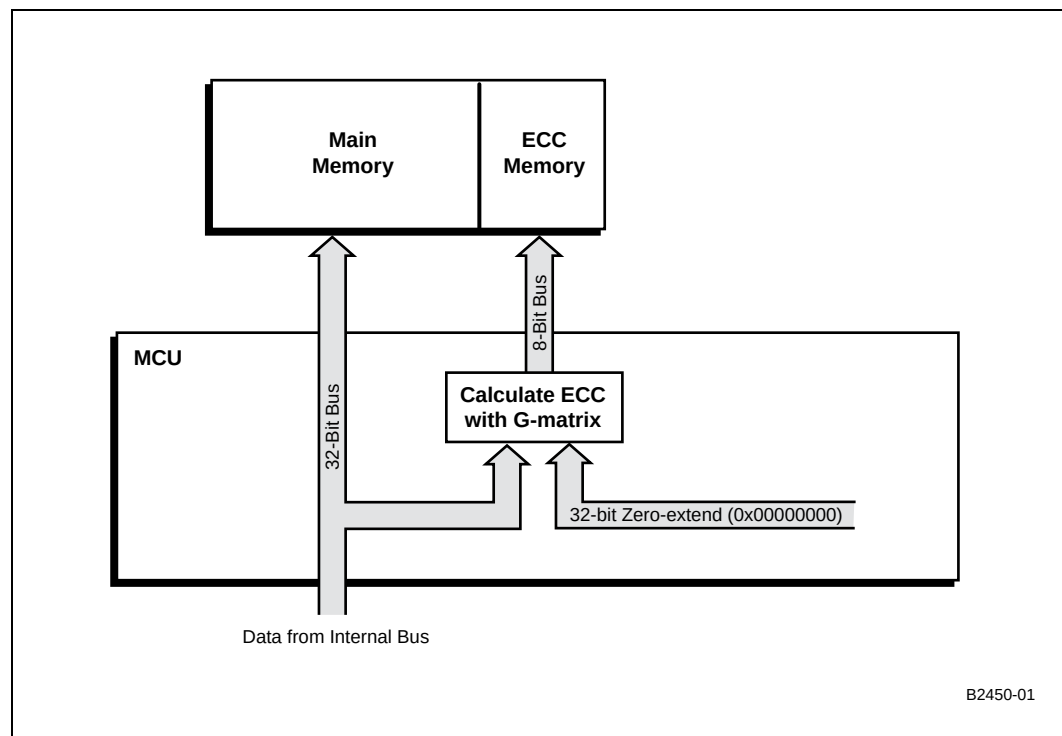
The algorithm for a write transaction is:

```

if data to write is 64/32 bits wide
    Generate the ECC_with the G-matrix
    Write the new data and ECC
else {Partial Write}
    Read entire 64/32-bit data word from memory
    Merge the new data portion with the data from memory
    Generate the new ECC with the G-matrix
    Write new data and ECC
  
```

Figure 118 shows how the data logically flows through the ECC hardware for a write transaction.

Figure 118. ECC Write Flow





The G-Matrix in [Figure 119](#) generates the ECC. The data to be written is input to the matrix and the output is the ECC code. Each row of the G-Matrix indicates which data bits of DATA[63:0] needs to be XORED together to form the ECC bit. The resulting ECC bits are driven on DDRI_CB[7:0].



11.2.3.2 ECC Generation for Partial Writes

Figure 119. IXP45X/ IXP46X product line G-Matrix (Generates the ECC)

Data Bit Positions

	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
CB0																																
CB1																																
CB2																																
CB3																																

CB4																																
CB5																																
CB6																																
CB7																																

Data Bit Positions

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB0																																
CB1																																
CB2																																
CB3																																

CB4																																
CB5																																
CB6																																
CB7																																



If the memory transaction writes less than the data bus width programmed in the SDCR, then the DDRI SDRAM Control Block translates the write transaction into a read-modify-write transaction. For a partial write, the DDRI SDRAM Control Block calculates the ECC for the modified datum and writes it back. If an external unit issues a write cycle with partial data to an MCU port, the MCU:

1. Issues a 32-bit read.
2. Modifies the value with the new portion to be written.
3. Calculates the ECC on the modified value.
4. Writes the 32-bit value and ECC.

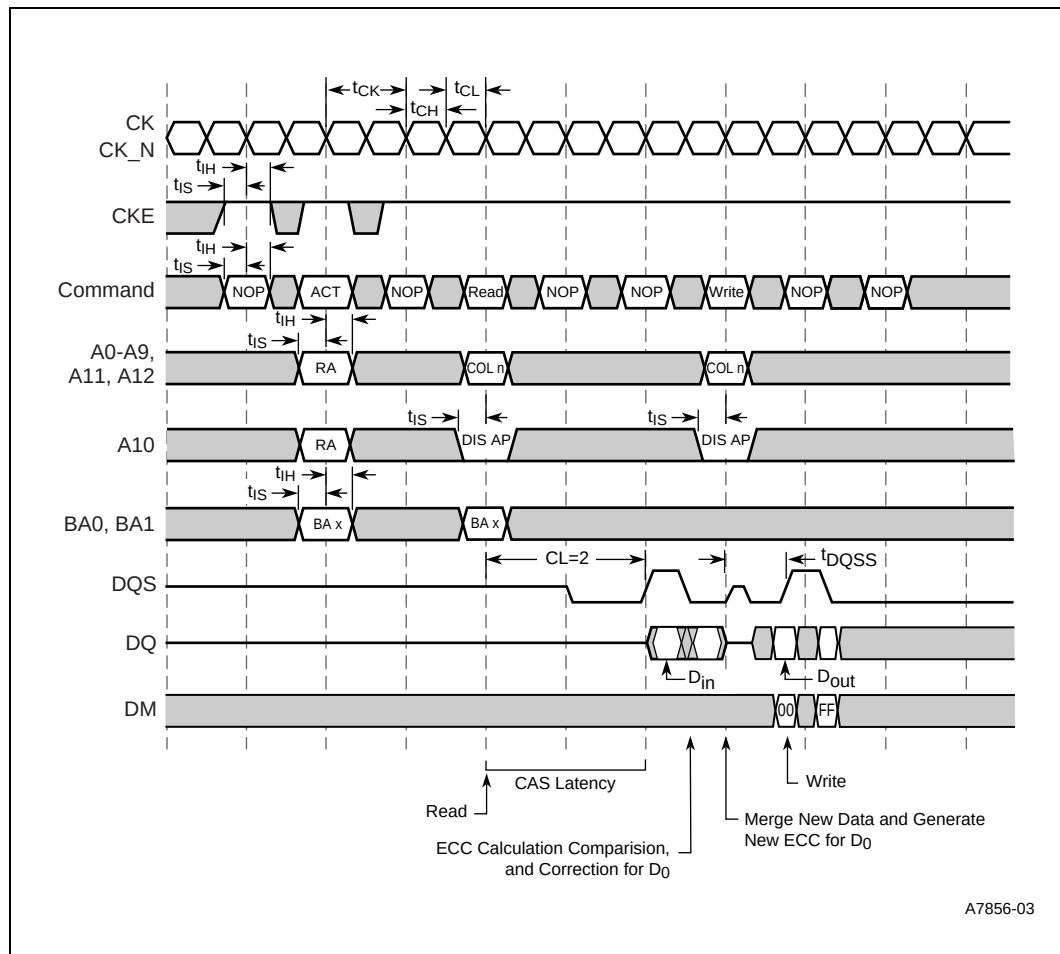
Note: If the MCU detects a single-bit error during the read, it is corrected BEFORE being merged with the write data so *the corrected data is written back to the array*. If a multi-bit error is detected, the MCU causes an interrupt to the core by writing to the MCISR. The memory location is overwritten by the MCU with the error data but valid ECC, making the contents of memory invalid. For more details on how the MCU handles error conditions, see [“Interrupts/Error Conditions” on page 627](#).

Figure 120 shows an example where the data of a write transaction is less than 64-bits wide. The waveform illustrates how the DDRI SDRAM Control Block issues a read-modify-write cycle for the data (D_1).

Note: In 32-bit wide memory, the DDRI SDRAM Control Block will still generate 8-bit wide ECC by zero extending the data to 64-bits. A partial write is a write of less than 4 Bytes.

Note: An undefined INCR read from the AHB bus will cause the MCU to read an entire cache line (8 words). If ECC error correction and detection is enabled, it is necessary to initialize a complete cache line before performing an undefined INCR read on any words or bytes within that cache line. If the entire cache line is not initialized and an undefined INCR read occurs, any bytes or words that were not initialized will be read, potentially causing an erroneous ECC error.

Note: ECC must be enabled or disabled in the ECCR MMR before any transaction to DDRI occurs, and its state must NOT subsequently be changed after DDRI transactions have begun.

Figure 120. Sub 64-bit DDRI SDRAM Write (D_0)



11.2.3.3 ECC Checking

If enabled, the ECC logic uses the following ECC read algorithm. This algorithm corrects the data before it is driven onto the internal bus.

The ECC algorithm for a read transaction is:

```
Read 64/32-bit data and 8-bit ECC (For the IXP45X/IXP46X network processors, 32 bit
read data is zero extended to 64 bits)

Compute the syndrome by passing the 64-bit data through the G-Matrix and XORing
the 8-bit result with the 8-bit ECC

if the syndrome <> 0 {ECC Error}

    Look up in H-matrix to determine error type
    Register the address where the error occurred
    if error is correctable {single bit}
        if single-bit error correction is enabled
            Correct data
            Send corrected data to internal bus
        if single bit error reporting is enabled
            Interrupt core for software scrubbing
    else {uncorrectable}
        if the read cycle is not part of a RMW cycle {read}
            Target-Abort the Internal Bus read transaction.
        else {write requiring RMW}
            Merge the new data portion with the read data from memory
            Generate the new ECC with the G-matrix
            Write new data and ECC
        if multi-bit error reporting is enabled
            Interrupt the core for uncorrectable error
```

When the MCU reads the ECC code from the memory subsystem, it is compared (XORed) with an ECC that the MCU generates from the data read from the memory. The result is called the syndrome. [Table 213](#) shows how the MCU decodes the syndrome for DDR1 SDRAM read cycles.

Table 213. Syndrome Decoding

Error Type	Symptom
None	The syndrome is 0000 0000.
Single-Bit	Use the H-Matrix in Figure 122 to determine which bit the MCU will invert to fix the error.
Multi-Bit	If the Syndrome does not match an 8-bit value in the H-matrix, the error is uncorrectable.

[Figure 121](#) shows how the data flows through the ECC hardware for a read transaction.



Figure 121. ECC Read Data Flow

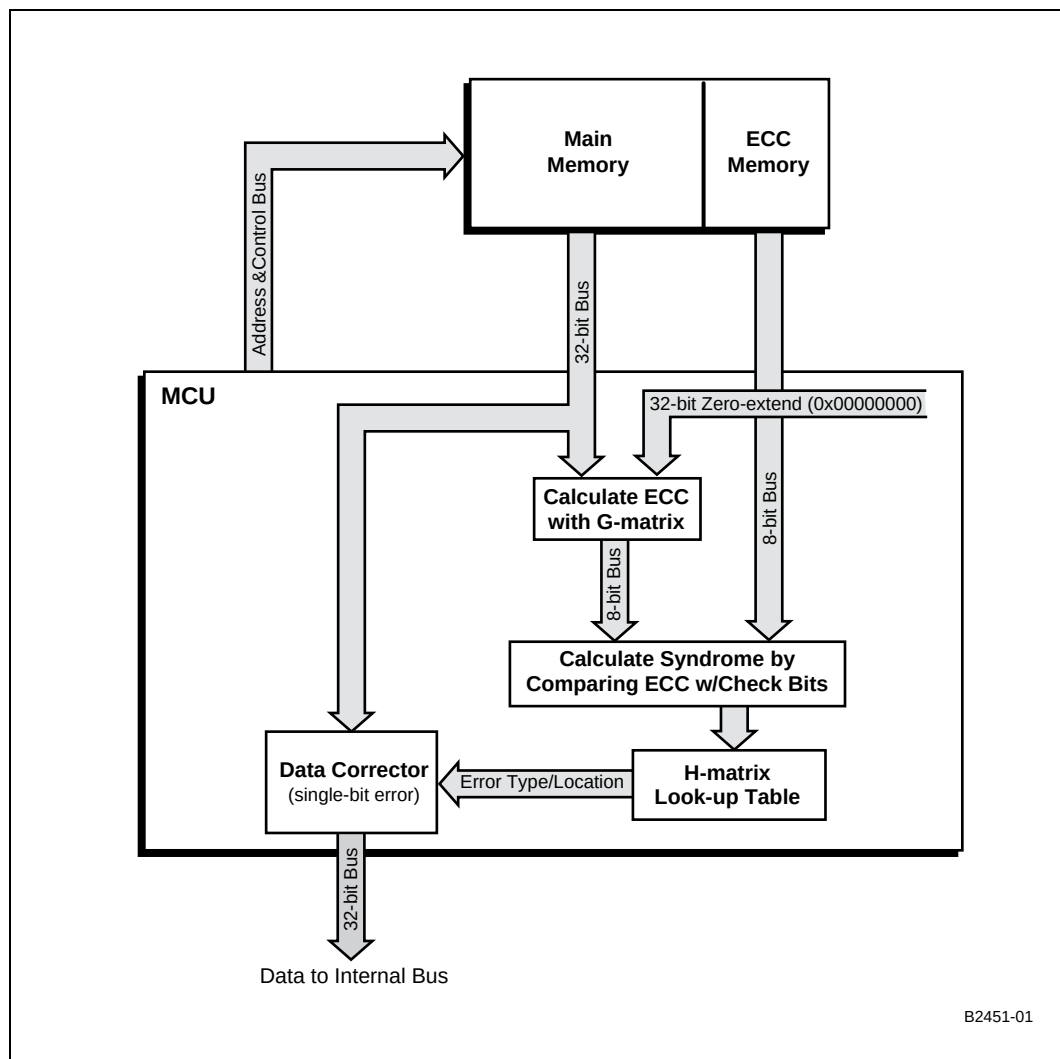


Figure 122 illustrates the H-Matrix used for decoding the syndrome. For single-bit errors, the H-Matrix indicates the bit that contains the error and, consequently, which bit to fix.



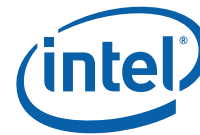


**Figure 122. Intel® IXP45X and Intel® IXP46X Product Line of Network Processors H-Matrix
(Indicates the Single-Bit Error Location)**

Bit Positions

	CB 0	CB 1	CB 2	CB 3	CB 4	CB 5	CB 6	CB 7	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
S0	1									1			1	1		1	1	1	1	1	1	1	1	1					1	1	1					
S1		1							1	1				1		1	1	1	1	1				1	1	1	1	1			1	1	1	1	1	
S2			1						1				1	1	1	1				1	1	1	1	1	1	1	1	1								
S3				1					1				1			1														1	1	1	1	1	1	1

Bit Positions

	CB 0	CB 1	CB 2	CB 3	CB 4	CB 5	CB 6	CB 7	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
S4					1				1	1	1				1				1					1					1						1	
S5						1			1	1	1				1				1			1					1				1			1		
S6							1		1	1	1				1				1			1				1				1			1			
S7								1	1	1	1				1	1	1				1			1					1				1			

Bit Positions

	CB 0	CB 1	CB 2	CB 3	CB 4	CB 5	CB 6	CB 7	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
S0					1	1	1								1					1										1					1	
S1						1	1								1				1									1							1	
S2	1	1	1	1																																
S3	1	1	1	1																																

Bit Positions

	CB 0	CB 1	CB 2	CB 3	CB 4	CB 5	CB 6	CB 7	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
S4					1				1	1					1				1																	
S5						1			1	1					1																					
S6							1		1	1					1																					
S7								1	1	1					1	1	1																			



Referring to [Figure 121](#), the syndrome bits are created by XORing the data bits as indicated by the appropriate row of the G-Matrix in [Figure 119](#) with the corresponding ECC bit. For example, the MCU derives syndrome bit 0 by XORing data bits 0, 4, 8, 12, 16, 20, 25, 29, 31, 40, 43, 48, 56, 58, 59, 62, and ECC bit 0 (physically read on DDRI_CB[7:0]). The MCU performs eight such XOR operations (one per syndrome bit).

If decoding the syndrome indicates multi-bit error (see [Table 213](#)), the transaction results in a target-error for Internal Bus transactions, or a multi-bit error in the BIU for Core transactions. If an internal bus master detects a target-abort, the master asserts an interrupt to the core. Write cycles are posted to the memory transaction queues, and already completed to the initiating master. For write cycles with a multi-bit error and with ECC Error reporting enabled, the MCU reports the interrupt in the MCISR and interrupts the core.

If the syndrome indicates a single-bit error and single-bit error correction is enabled, the H-Matrix is used to determine the bit in error (see [Figure 122](#)). For example, if the syndrome was 1100 0001, the error is with bit 0 of DDRI_DQ[31:0]. The MCU inverts bit 0 before driving the data on DATA[63:0].

If error reporting is enabled in the ECCR and the MCU detects a single-bit or multi-bit error, the MCU stores the address in ECARx and the syndrome in ELOGx. Then, the MCU signals an interrupt to the core. Software decides how to proceed through an interrupt handler. By registering the address in ECARx, software can identify the faulty DIMM.

For details about the MCU error conditions and how the MMR registers are affected, refer to ["Interrupts/Error Conditions" on page 627](#).

Note: In 32-bit wide memory, the DDRI SDRAM Control Block will still generate 8-bit wide ECC by zero-extending the data to 64-bits. A partial write is a write of less than 4 Bytes.

11.2.3.4 Scrubbing

Fixing the data error in memory is called scrubbing. The IXP45X/IXP46X network processors rely on Intel XScale processor software to perform the scrubbing. When the MCU detects an error during a read, the MCU logs the address where the error occurred and interrupts the Intel XScale processor. The Intel XScale processor decides how to fix the error through an interrupt handler. Software could decide to perform the scrubbing on:

- the data location that failed
- the entire row of the data that failed
- the entire memory

For single-bit errors reported on a write transaction scrubbing is not required, as the MCU will have scrubbed the data during the RMW operation. For single-bit errors, the error is fixed by reading the location that failed and writing back the data after the ECC hardware fixed it. The scrubbing routine should read either DWORD of the 64-bit memory space (QWORD aligned location) using an **ld** instruction and write the data back with an **st** instruction. Software should isolate activity on the memory location to guarantee atomicity.

Note: If the scrubbing routine reads the failed location in order to fix the single-bit error, a second error will be reported. Therefore, software should disable single-bit ECC reporting during the scrubbing routine. Also, the scrubbing routine should be aware that partial writes will automatically scrub the QWORD aligned location if it contains a single-bit ECC error.

Multi-bit errors cannot be fixed by the H-Matrix.



11.2.3.4.1 ECC Example Using the H-Matrix

Assume the Intel XScale processor writes 1234 5678 9ABC DEF0H to the DDR1 SDRAM memory space. The Intel XScale processor Address Decoder decodes the address and determines the write should be sent to the Core Memory Transaction Queue. The CMTQ latches the transaction with data 1234 5678 9ABC DEF0H on DATA[63:0].

During the next CMTQ tenure, this transaction is processed and the DDR1 SDRAM Control Block receives the data and then must calculate the ECC code.

Using the G-Matrix in [Figure 119](#), the DDR1 SDRAM Control Block creates each check bit by XORing the appropriate bits in the row. Using 1234 5678 9ABC DEF0H, the ECC code generated is D2H. This code is written with the data to the SDRAM devices on DDR1_CB[7:0].

Assume that bit 17 was corrupted in the array. Therefore, the bit has been inverted from 0 to 1.

At some later point in time, the core wishes to read from the same address. The Intel XScale processor issues a read transaction which is latched by the CMTQ after the Core Address Decoder decodes the address and determines the read targets the DDR1 SDRAM address space. Upon the receipt of 1234 5678 9ABE DEF0H on DDR1_DQ[31:0], the DDR1 SDRAM Control Block calculates the syndrome with the G-Matrix in [Figure 119](#). The DDR1 SDRAM Control Block calculates a syndrome of 52H.

Note: During a memory write, ECC code is created by XORing the appropriate data bits indicated by the G-Matrix. The syndrome is created during a memory read by XORing the 8-bit value generated by XORing appropriate data bits on DDR1_DQ[31:0] indicated by the G-Matrix with the check bits DDR1_CB[7:0].

Referring to [Table 213](#), if the syndrome is non-zero and matches a value in the H-Matrix, there is a single-bit error that can be fixed. A syndrome of 52H matches a value in the H-Matrix (see [Figure 122](#)) which indicates that bit 17 has an error. The DDR1 SDRAM Control Block inverts bit 17 prior to returning the corrected data on DATA[63:0]. The MCU returns 1234 5678 9ABC DEF0H on DATA[63:0].

Assuming this was the first error, the MCU records the address where the error occurred in ECAR0 and error type in ELOG0. If error reporting is enabled in the ECCR, the MCU writes a 1 to MCISR[0] which generates an interrupt to the core. A software interrupt handler scrubs the array and fixes the error in bit 17. Unless more errors occur, future reads from this location do not result in an error.

11.2.3.5 ECC Disabled

If software disables ECC, the MCU does not check the ECC byte during a read access. However, the MCU does generate the ECC byte during a write access. If ECC is enabled but the access is less than 32 bits, the MCU does not check, modify, nor write to the ECC byte.

11.2.3.6 ECC Testing

[Section 11.2.3.4, “Scrubbing”](#) explains how the software is responsible for correcting an error in the memory array once it has been detected by the ECC logic. The MCU implements the ECTST register providing the programmer the ability to test error handling software. For write transactions, the ECTST register value is XORed with the generated ECC. This inverts the bits where the mask is set prior to writing the ECC to memory. When the MCU reads the address later, the ECC mismatches and the error condition occurs (see [“Interrupts/Error Conditions” on page 627](#)).



11.2.4 Overlapping Memory Regions

The MCU supports two independent memory regions:

- Memory Mapped Register (MMR) Space
- DDRI SDRAM Memory Space

The MMR memory space is fixed at CC00 E000H to CC00 FFFFH. Software programs the DDRI SDRAM memory space by providing a base address in SDBR, each of the two bank boundaries in SBR0 and SBR1.

For the IXP45X/IXP46X network processors, the two memory regions must never be programmed to overlap.

11.2.5 DDRI SDRAM Clocking

The MCU provides 6 clocks to the DDRI SDRAM memory subsystem at the selected DDRI SDRAM command rate. There are three positive clocks DDRI_CK[2:0] and three negative clocks DDRI_CK_N[2:0]. The 72-bit 2-bank unbuffered *JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79*, June 2004 requires 6 clocks to distribute the loading across eighteen x8 DDRI SDRAM components.

Note: For IXP45X/IXP46X network processors, there will be a maximum of 10 chips (two banks of x8 DRAM chips for a 40-bit bus).

11.2.6 Performance Monitoring

By setting up the system level PMU registers for the IXP45X/IXP46X network processors, the following parameters can be monitored.

- Page hit and page miss counts for each of the 8 possible open pages of DDRI SDRAM.
- AHB bus Read latency (from request to valid data back for each of the AHB bus ports).
- Core memory bus read latency (from the Intel XScale processor to DDRI request to valid data back for each of the eight outstanding read requests possible from the Intel XScale processor to memory).
- The PMU status comes into the PMU event counters in the form of three different buses. These are:
 - DDRI_PMU_EVENT[4:0] where bit 4 = valid, bit 3 = hit/miss, bits 2:0 are the page number
 - BIU_PMU_RDACTIVE[7:0] where each bit is the enable for an active read in progress
 - DDRI_PMU_RDACTIVE[1:0] where bit 1 = South AHB read active and bit 0 = North AHB read active



11.3 Power Failure Mode

This mode is not supported by the IXP45X/IXP46X network processors as there is no support for self-refresh.

11.4 Interrupts/Error Conditions

The MCU has two conditions which require intervention from the Intel XScale processor. If a single-bit error is detected during a read cycle, the MCU can correct the data returned but software needs to fix the error in the memory array. If a multi-bit error is detected, the core decides how to handle the condition. For all ECC errors, the MCU records the requester of the transaction resulting in the error in ELOGx[23:16] and interrupts the core.

If the MCU detects an ECC error during a read or write cycle¹, MCISR[0] or MCISR[1] is set to 1. Whenever the MCU toggles one of the MCISR bits from 0 to 1, an interrupt is generated to the core.

Table 214 shows how the MCU responds to error conditions.

Table 214. MCU Error Response

Error Type	MCU Action ¹
Single-Bit during a read or write	Fix Error (if ECC error correction is enabled in the ECCR)
Multi-bit during a read	Interrupt the Intel XScale® Processor through the Interrupt Controller or Terminate the Core transaction, notify the BIU of multi-bit error
Multi-bit during a write	New ECC is generated with bad data and written to DDRI SDRAM array. Data location is no longer valid.

1. The ECC Enable bit in the ECCR needs to be set in order for these actions to occur.

Note: If ECC reporting is enabled with ECCR[1] or ECCR[0] and an ECC error occurs, MCISR[1] or MCISR[0] is set and ELOGx/ECARx logs the error in addition to the actions in Table 214.

1. Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See “ECC Generation” on page 615 for details.



11.4.1 Single-Bit Error Detection

When enabled, the MCU interrupts the core when the ECC logic detects a single-bit error by setting the appropriate bit in the MCISR register. The Intel XScale processor knows the interrupt was caused by a single-bit error by polling the ELOG0 or ELOG1 register. The DDR1 SDRAM Control Block ensures that correct data is returned but the interrupt handler is responsible for scrubbing the error in the array (refer to [“Scrubbing” on page 624](#)).

An example flow for a single-bit error with error detection and reporting enabled is:

- A single-bit ECC error is detected on the data bus by the MCU.
- The MCU fixes the error prior to returning the data.
- The MCU clears ELOG0[8] indicating a single-bit error.
- The MCU records the requester of the transaction that resulted in an error in ELOG0[23:16]
- The MCU loads ELOG0[7:0] with the syndrome that indicated the error.
- The MCU loads ECAR0[31:2] with the address where the error occurred.
- Since the Intel XScale processor needs to scrub the error in the array, the MCU sets MCISR[0] to 1 (assuming it is not already set).
 - Setting any bit in the MCISR causes an interrupt to the Intel XScale processor.
- Software polls the interrupt status register. Bit 0 set to 1 indicates that the first error has occurred.
- Software polls ELOG0 and ECAR0 and scrubs the error at the location specified by ECAR0.
- Software writes a 1 to MCISR[0] thereby clearing it.

If software does not perform error scrubbing, the probability of an unrecoverable multi-bit error increases for the memory location containing the single-bit error.

ECARx and ELOGx remain registered until software explicitly clears them.

If a second error occurs before software clears the first by resetting MCISR[0] or MCISR[1], the error is recorded in the remaining ELOGx/ECARx register. If none are available, the error is not logged but the MCU carries out the action described in [Table 214](#).



11.4.2 Multi-Bit Error Detection

If a multi-bit error occurs during a read or write transaction and error reporting is enabled, the MCU sets MCISR[0] or MCISR[1] which asserts an interrupt to the core. Upon receiving an interrupt, the core knows the interrupt was caused by a multi-bit error by polling the ELOGx registers.

When MCU detects a multi-bit error during a read cycle and ECC calculation is enabled in the ECCR, the MCU target aborts the transaction in the IBMTQ, indicating to the internal bus masters that an unrecoverable error has been detected. For core transactions issued by the CMTQ, when a multi-bit error is detected during a read cycle, the MCU signals a multi-bit error to the BIU. The MCU records the error type in ELOGx and the address in ECARx.

When MCU detects a multi-bit error during a write¹ cycle and error reporting is enabled in the ECCR, the MCU records the first multi-bit error by programming ELOGx and ECARx. The MCU generates new ECC with the data before sending it on DDRI_DQ[31:0] so the contents of memory after the read-modify-write cycle will be corrupted with correct ECC.

If a second error occurs before software clears the first by resetting MCISR[0] or MCISR[1], the error is recorded in the remaining ELOGx/ECARx register. If none are available, the error is not logged but the MCU carries out the action described in [Table 214](#).

It is the interrupt handler's responsibility to decide how to handle this error condition and clear the MCISR.

11.5 Reset Conditions

Once RESET_IN_N is deasserted and 200 us have passed, software must issue the initialization sequence defined in [“DDRI SDRAM Initialization” on page 598](#). After initialization, the DDRI SDRAM devices are ready to be written to or read from. Reads issued prior to a write to the same address results in an ECC error and are not recommended if ECC is enabled.

While RESET_IN_N is asserted, the MCU initializes its MMR registers to the states defined in [“Register Definitions” on page 630](#).

1. Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See [“ECC Generation” on page 615](#) for details.



11.6 Register Definitions

A series of configuration registers control the MCU. Software can determine the status of the MCU by reading the status registers. [Table 216](#) summarizes all of the MCU registers and provides links to register details.

Note: When computing the register values based on JEDEC specifications, round your final answer up to the nearest integer value.

Note: Constant polling of MCU MMRs can result in inducing long latencies in peripheral unit DDRI SDRAM transactions, and therefore may negatively impact performance. Polling of MCU MMRs should be avoided.

Table 215. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 216. Memory Controller Register Table (Sheet 1 of 2)

Address	Register Name	Description	Reset Value	Page with Details
CC00 E500H	DDRI SDRAM Initialization Register - SDIR	DDRI SDRAM Initialization Register	0x0000 000FH	632
CC00 E504H	DDRI SDRAM Control Register 0 - SDCR0	DDRI SDRAM Control Register 0	0x0001 0004H	633
CC00 E508H	DDRI SDRAM Control Register 1 - SDCR1	DDRI SDRAM Control Register 1	0x0000 0000H	635
CC00 E50CH	SDRAM Base Register - SDBR	SDRAM Base Register	0x0001 0004H	637
CC00 E510H	SDRAM Boundary Register 0 - SBR0	DDRI SDRAM Boundary Register 0	0x0000 0000H	638
CC00 E514H	SDRAM Boundary Register - SBR1	DDRI SDRAM Boundary Register 1	0x0000 0000H	639
CC00 E51CH	ECC Control Register - ECCR	ECC Control Register	0x0000 0000H	640
CC00 E520H CC00 E524H	ECC Log Registers - ELOG0, ELOG1	ECC Log Registers	0x0000 0000H	641
CC00 E528H CC00 E52CH	ECC Address Registers - ECAR0, ECAR1	ECC Address Registers	0x0000 0000H	642
CC00 E530H	ECC Test Register - ECTST	ECC Test Register	0x0000 0000H	643
CC00 E534H	Memory Controller Interrupt Status Register - MCISR	Memory Controller Interrupt Status Register	0x0000 0000H	644
CC00 E53CH	MCU Port Transaction Count Register - MPTCR	MCU Port Transaction Count Register	0x0000 0011H	645

**Table 216. Memory Controller Register Table (Sheet 2 of 2)**

Address	Register Name	Description	Reset Value	Page with Details
CC00 E540H	MCU Preemption Control Register - MPCR	MCU Preemption Control Register	0x0000 0000H	646
CC00 E548H	Refresh Frequency Register - RFR	Refresh Frequency Register	0x0000 0000H	646
0 — CC00 E550H 1 — CC00 E554H 2 — CC00 E558H 3 — CC00 E55CH 4 — CC00 E560H 5 — CC00 E564H 6 — CC00 E568H 7 — CC00 E56CH	SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7	SDRAM Page Registers	0x0000 0000H	647



11.6.1 DDRI SDRAM Initialization Register SDIR

The DDRI SDRAM Initialization Register (SDIR) is responsible for programming the operation of the DDRI SDRAM device state machines. The SDIR provides a method for software to execute the DDRI SDRAM initialization sequence (see “[DDRI SDRAM Initialization](#)” on page 598).

Register Name:	DDRI SDRAM Initialization Register - SDIR																										
Hex Offset Address:	CC00 E500H												Reset Hex Value:	0x0000 000FH													
Register Description:	DDRI SDRAM Initialization Register																										
Access: See below.																											
31																							4	3			0
(Reserved)																											

Register		DDRI SDRAM Initialization Register - SDIR		
Bits	Name	Description	Default	Access
31:0 4	(Reserved)		0	RO
03:0 0		Special DDRI SDRAM Command: These bits are used for DDRI SDRAM initialization. See “DDRI SDRAM Initialization” on page 598 for details. While not in the initialization sequence, these bits must be set to 1xx₂. For details on the exact DDRI SDRAM commands, refer to Table 211, “DDRI SDRAM Commands” on page 598. 0000₂ - Mode-Register-Set Command where DLL is not Reset and CAS Latency as specified in SDCR0 0001₂ - Mode-Register-Set Command where DLL is Reset and CAS Latency and Burst Length of four 0010₂ - Precharge-All Command: The MCU issues one precharge-all command to the DDRI SDRAM devices. 0011₂ - NOP Command: The MCU issues one NOP command to the DDRI SDRAM devices. 0100₂ - Extended Mode-Register Set (EMRS) Command where DLL is enabled on the DDRI SDRAM devices. 0101₂ - Extended Mode-Register Set (EMRS) Command where DLL is disabled on the DDRI SDRAM devices. 0110₂ - Auto-Refresh Command: The MCU issues one refresh sequence (i.e. precharge-all followed by auto-refresh command) to the DDRI SDRAM devices. - Others - Normal DDRI SDRAM Operation	1111 ₂	RW



11.6.2 DDRI SDRAM Control Register 0 SDCR0

The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDRI SDRAM state machines. The SDCR0 specifies the DIMM type, data bus width, and some SDRAM timing parameters required by the DDRI SDRAM state machine as defined in “[DDRI SDRAM Initialization](#)” on page 598 and “[DDRI SDRAM Mode Programming](#)” on page 602. The remaining SDRAM timing parameters required by the DDRI SDRAM state machine are set in SDCR1.

Note: Each parameter field contains either an IXP45X/IXP46X network processors-required value limit or an example value derived from a typical device datasheet.

Register Name:				DDRI SDRAM Control Register 0 - SDCR0																																					
Hex Offset Address:				CC00 E504H								Reset Hex Value:				0x0001 0004H																									
Register Description:				DDRI SDRAM Control Register 0																																					
Access: See below.																																									
31				28	27	26			24	23	22			20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00							
					(Rsvd)						(Rsvd)						(Rsvd)						(Rsvd)						(Rsvd)						(Rsvd)						(Rsvd)

Register		DDRI SDRAM Control Register 0 - SDCR0 (Sheet 1 of 2)		
Bits	Name	Description	Default	Access
31:2 8	RAS	RAS: Active to Precharge duration in MCLK periods. This value is computed by using the t_{RAS} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 3. $RAS = t_{RAS} - 1$ NOTE: t_{RAS} must be converted to mclks for this equation. For example, a device specification yields that t_{RAS} is 40ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{RAS} = (40ns / 7.5ns) = 5.33$, round up to 6 mclks. Thus, $RAS = 6 - 1 = 5$, so a value of 0101 ₂ should be programmed.	0H	RW
27	(Reserved)		0 ₂	RO
26:2 4	RP	RP: Precharge Command Period in MCLK periods. This value is computed by using the t_{RP} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 4. $RP = t_{RP} - 1$ NOTE: t_{RP} must be converted to mclks for this equation. For example, a device specification yields that t_{RP} is 20ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{RP} = (20ns / 7.5ns) = 2.67$, round up to 3 mclks. Thus, $RP = 3 - 1 = 2$, so a value of 010 ₂ should be programmed.	000 ₂	RW
23	(Reserved)		0 ₂	RO



Register		DDR1 SDRAM Control Register 0 - SDCR0 (Sheet 2 of 2)		
Bits	Name	Description	Default	Access
22:2 0	RCD	RCD: Active to Read or Write Delay in MCLK periods. This value is computed by using the t_{RCD} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 5. $RCD = t_{RCD} - 1$ NOTE: t_{RCD} must be converted to mclks for this equation. For example, a device specification yields that t_{RCD} is 20ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{RCD} = (20ns / 7.5ns) = 2.67$, round up to 3 mclks. Thus, $RCD = 3 - 1 = 2$, so a value of 010_2 should be programmed.	000_2	RW
19:1 8	(Reserved)		00_2	RO
17:1 6	EDP	EDP: Data Path Latency in MCLK periods. A value of 10_2 should be programmed for the IXP45X/IXP46X network processors.	01_2	RW
15:1 4	(Reserved)		00_2	RO
13:1 2	WDL	WDL: Write Data Latency in MCLK periods. A value of 00_2 must be programmed for the IXP45X/IXP46X network processors.	00_2	RW
11:1 0	(Reserved)		00_2	RO
09:0 8	CAS	CAS: Indicates the CAS Latency of the memory device (provided in the JEDEC/Vendor specification). 00 = 2 MCLK periods 01 = 2.5 MCLK periods 10 = RESERVED 11 = RESERVED	00_2	RW
07:0 6	(Reserved)		00_2	RO
05:0 4	(Reserved)	RESERVED for DDRII. A value of 00_2 must be programmed for the IXP45X/IXP46X network processors.	00_2	RW
03	(Reserved)		0_2	RO
02	DDR	DDR: Identifies the generation of DDR SDRAM selected by the MEM_TYPE reset strap. A MEM_TYPE value of 1_2 must be used for the IXP45X/IXP46X network processors. 0 = RESERVED for DDRII/400MHz 1 = DDRI/266MHz	Varies with external state of MEM_TYPE at reset. Should default to 1	RO
01	Data Bus Width	Data Bus Width: Indicates the width of the data bus. See Section 11.2.2.5, "32-Bit Data Bus Width" on page 594. A value of 1_2 must be used for the IXP45X/IXP46X network processors. 0 = RESERVED for 64 bits 1 = 32 bits	0_2	RW
00	(Reserved)	RESERVED for DIMM Type. A value of 0_2 must be used for the IXP45X/IXP46X network processors.	0_2	RO



11.6.3 DDRI SDRAM Control Register 1 SDCR1

The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDRI SDRAM state machines as defined in [Section 11.2.2.8, “DDRI SDRAM Initialization”](#) on page 598 and [Section 11.2.2.9, “DDRI SDRAM Mode Programming”](#) on page 602. The SDCR1 specifies the remaining SDRAM timing parameters required by the DDRI SDRAM state machine not specified in SDCR0.

Note: Each parameter field contains either an IXP45X/IXP46X network processors-required value limit or an example value derived from a typical device datasheet.

Register Name:		DDRI SDRAM Control Register 1 - SDCR1																																							
Hex Offset Address:		CC00 E508H								Reset Hex Value:		0x0000 0000H																													
Register Description:		DDRI SDRAM Control Register 1																																							
Access: See below.																																									
31	30			28	27				24	23	22			20	19			17	16					12	11			09	08					04	03	02			00		
(Rsvd)										(Rsvd)				(Rsvd)																								R			

Register		DDRi SDRAM Control Register 1 - SDCR1 (External Version) (Sheet 1 of 2)		
Bits	Name	Description	Default	Access
31	(Reserved)	RESERVED for DDRI. A value of 0 ₂ must be used for the IXP45X/IXP46X network processors.	0 ₂	RW
30:2 8	RTCMD	RTCMD : Read to non-Write Command turnaround period in MCLK periods. A value of 010 ₂ must be programmed for the IXP45X/IXP46X network processors.	000 ₂	RW
27:2 4	WTCMD	WTCMD: Write to Command (non-Read) turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS}) and Write Recovery (t_{WR}) measured for the device (provided in the JEDEC/ Vendor specification) using the following equation: Equation 6. $WTCMD = t_{CAS} + t_{WR}$ NOTE: t_{CAS} and t_{WR} must be converted to mclks for this equation. For example, a device specification yields that t_{CAS} is 2.5 mclks and t_{WR} is 15ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{WR} = (15ns / 7.5ns) = 2$ mclks. Thus, $WTCMD = 2.5 + 2 = 4.5$, round up to 5 mclks, so a value of 0101₂ should be programmed.	000 ₂	RW
23	(Reserved)		0 ₂	RO



Register		DDR1 SDRAM Control Register 1 - SDCR1 (External Version) (Sheet 2 of 2)		
Bits	Name	Description	Default	Access
22:2 0	RTW	RTW: Read to Write turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS}) for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 7. $RTW = t_{CAS} + EDP + 1$ NOTE: t_{CAS} must be converted to mclks for this equation. A programmed value of 000_2 represents a value of 8_{10}. For example, a device specification yields that t_{CAS} is 2.5 mclks and the suggested EDP value of 2 mclks is used. Thus, $RTW = 2.5 + 2 + 1 = 5.5$, round up to 6 mclks, so a value of 0110_2 should be programmed. NOTE: The MCU allows for back-to-back reads, so long as they are to open pages.	000_2	RW
19	(Reserved)	RESERVED. A value of 0_2 must be used for the IXP45X/IXP46X network processors.	0_2	RW
18:1 7	(Reserved)			RO
16:1 2	RFC	RFC: Auto Refresh Command Period in MCLK periods. This value is computed by using the t_{RFC} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 8. $RFC = t_{RFC} - 1$ NOTE: t_{RFC} must be converted to mclks for this equation. For example, a device specification yields that t_{RFC} is 120ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{RFC} = (120ns / 7.5ns) = 16$ mclks. Thus, $RFC = 16 - 1 = 15$, so a value of 01111_2 should be programmed.	$0\ 0000_2$	RW
11:0 9	(Reserved)	RESERVED. A value of 000_2 must be used for the IXP45X/IXP46X network processors.	000_2	RW
08:0 4	RC	RC: Active to Active and Active to Auto Refresh command period in MCLK periods. This value is computed by using the t_{RC} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 9. $RC = t_{RC} - 1$ NOTE: t_{RC} must be converted to mclks for this equation. For example, a device specification yields that t_{RC} is 65ns (and the device speed is 133MHz, so the mclk period is 7.5ns). $t_{RC} = (65ns / 7.5ns) = 8.67$, round up to 9 mclks. Thus, $RC = 9 - 1 = 8$, so a value of 01000_2 should be programmed.	$0\ 0000_2$	RW
03	(Reserved)			
02:0 0	WTRD	WTRD: Write-to-Read turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS}) and internal Write to Read command delay (t_{WTR}) measured for the device (provided in the JEDEC/Vendor specification) using the following equation: Equation 10. $WTRD = t_{CAS} + t_{WTR}$ NOTE: t_{CAS} and t_{WTR} must be converted to mclks for this equation. For example, a device specification yields that t_{CAS} is 2.5 mclks and t_{WTR} is 1 mclk. Thus, $WTRD = 2.5 + 1 = 3.5$, round up to 4 mclks, so a value of 100_2 should be programmed.	000_2	RW



11.6.4 DDRI SDRAM Base Register SDBR

This register indicates the beginning of SDRAM space. See [Section 11.2.2.2, “DDRI SDRAM Bank Sizes and Configurations” on page 590](#) for usage details. There can be two contiguous physical banks defined by SBR0 and SBR1 in the DDRI SDRAM subsystem starting at this address.

Note: DDRI SDRAM memory space must **never** cross a 2 Gbyte boundary.

Note: This register should be read back after being written, before the Intel XScale processor performs transactions which address the DDRI SDRAM.

Register Name:		SDRAM Base Register - SDBR																												
Hex Offset Address:		CC00 E50CH										Reset Hex Value:		0x0000 0000H																
Register Description:		SDRAM Base Register																												
Access: See below.																														
31							25	24																						00
		(Reserved)																												

Register		SDRAM Base Register - SDBR		
Bits	Name	Description	Default	Access
31:2 5		SDRAM Base Address: These bits define the upper seven bits of the DDRI SDRAM base address. These seven bits are compared with ADDR[31:25] to determine if the internal bus transaction hits SDRAM memory space. See Table 205 .	00H	RW
24:0 0	(Reserved)		000000H	RO



11.6.5 DDRI SDRAM Boundary Register 0 SBR0

This register indicates the upper boundary of SDRAM bank 0 and its memory technology. If bank 0 is unpopulated, SBR0[6:0] is programmed either with all zeros or a number equal to the value in SDBR[31:25]. See “[DDRI SDRAM Bank Sizes and Configurations](#)” on page 590 for more details and programming examples.

Note: DDRI SDRAM memory space must **never** cross a 2-Gbyte boundary.

Note: This register should be read back after being written, before the Intel XScale processor performs transactions which address the DDRI SDRAM.

Register Name:		SDRAM Boundary Register 0 - SBR0																					
Hex Offset Address:		CC00 E510H				Reset Hex Value:		0x0000 0000H															
Register Description:		DDR1 SDRAM Boundary Register 0																					
Access: See below.																							
31	30	29														07	06						00
		(Reserved)																					

Register		SDRAM Boundary Register 0 - SBR0		
Bits	Name	Description	Default	Access
31:3 0		SDRAM Technology: Defines the memory subsystem technology. • 00 128 Mbitx8/16, 256 Mbitx8, 512 Mbitx8/16, 1 Gbitx8 • 10 256 Mbitx16 • 11 1 Gbitx16 • 01 Reserved	00 ₂	RW
29:0 7	(Reserved)		0000000H	RO
06:0 0		SDRAM Boundary: Defines the upper limit of SDRAM bank 0. This value is compared with ADDR[31:25] to determine a bank hit or miss.	00000 ₂	RW



11.6.6 DDRI SDRAM Boundary Register 1 SBR1

This register indicates the upper boundary of SDRAM bank 1 and its memory technology. If bank 1 is unpopulated, SBR1[6:0] should be programmed to the value stored in SBR0[6:0]. If bank 1 is populated, SBR1[6:0] must be programmed greater than or equal to SBR0[6:0]. See ["DDRI SDRAM Bank Sizes and Configurations" on page 590](#) for more details and programming examples.

Note: DDRI SDRAM Memory Space must never cross a 2-Gbyte boundary.

Note: This register should be read back after being written, before the Intel XScale processor performs transactions which address the DDRI SDRAM.

Register Name:		SDRAM Boundary Register - SBR1																						
Hex Offset Address:		CC00 E514H				Reset Hex Value:		0x0000 0000H																
Register Description:		DDRI SDRAM Boundary Register 1																						
Access: See below.																								
31	30	29															07	06						00
		(Reserved)																						

Register		SDRAM Boundary Register - SBR1		
Bits	Name	Description	Default	Access
31:3 0		SDRAM Technology: Defines the memory subsystem technology. • 00 128 Mbitx8/16, 256 Mbitx8, 512 Mbitx8/16, 1 Gbitx8 • 10 256 Mbitx16 • 11 1 Gbitx16 • 01 Reserved	00 ₂	RW
29:0 7	(Reserved)		0000000H	RO
06:0 0		SDRAM Boundary: Defines the upper limit of DDRI SDRAM bank 1. This value is compared with ADDR[30:25] to determine a bank hit or miss.	000000 ₂	RW



11.6.7 ECC Control Register ECCR

This register programs the MCU error correction and detection capabilities. The configuration depends on the application's needs but a typical configuration is:

- ECC Mode Enabled
- Enable multi-bit error reporting
- Disable single-bit error reporting
- Enable single-bit error correcting

ECC must be enabled or disabled in the ECCR MMR before any transaction to DDR1 occurs, and its state must NOT subsequently be changed. For more details, see [“Error Correction and Detection” on page 614](#) and [“Interrupts/Error Conditions” on page 627](#).

Register Name:		ECC Control Register - ECCR																				
Hex Offset Address:		CC00 E51CH				Reset Hex Value:		0x0000 0000H														
Register Description:		ECC Control Register																				
Access: See below.																						
31																		04	03	02	01	00
(Reserved)																						

Register		ECC Control Register - ECCR			
Bits	Name	Description	Default	Access	
31:0 4	(Reserved)		000 0000H	RO	
03		ECC Enabled: Enables ECC Read Modify Write sequence for ECC calculation and generation during sub 64-bit writes. See “ECC Disabled” on page 625 . 0 = ECC Disabled (mode for Intel XScale processor ECC scrub) 1 = ECC Enabled (normal operation)	0 ₂	RW	
02		Single Bit Error Correction Enable: Enables or disables the correction of a single bit error. 0 = Disable single bit error correction 1 = Enable single bit error correction	0 ₂	RW	
01		Multi-Bit Error Reporting Enable: Enables or disables the reporting of a multi-bit error condition. 0 = Disable multi-bit error reporting 1 = Enable multi-bit error reporting	0 ₂	RW	
00		Single Bit Error Reporting Enable: Enables or disables the reporting of a single bit error condition. 0 = Disable single bit error reporting 1 = Enable single bit error reporting	0 ₂	RW	

Note: In order to ensure predictable ECC operation, Single Bit Error Correction Enable *must* be enabled whenever ECC is enabled. The reporting enables can be configured as desired.



11.6.8 ECC Log Registers ELOG0, ELOG1

The ECC Log Registers are responsible for logging the error types detected on the local memory bus. Two errors can be detected and logged. The error type is logged (single-bit or multi-bit) along with the syndrome that indicated the error. For a single-bit error, software can read this syndrome and determine which bit had the error in order to perform scrubbing. For a multi-bit error, the syndrome will not match an entry in the H-Matrix and thus, is uncorrectable (see Table 213, “Syndrome Decoding” on page 620).

The error recorded in ELOG0 corresponds to the address in ECAR0. ELOG1 corresponds to ECAR1.

The ELOGx registers comprise read-only bits and only have meaning if MCISR[0] or MCISR[1] is non-zero. For more details on error handling, see “Error Correction and Detection” on page 614.

Register Name:		ECC Log Registers - ELOG0, ELOG1																																		
Hex Offset Address:		CC00 E520H CC00 E524H						Reset Hex Value:				0x0000 0000H																								
Register Description:		ECC Log Registers																																		
Access: See below.																																				
31								24	23								16	15			13	12	11			09	08	07								0
(Reserved)																(Rsvd)						(Rsvd)														

Register		ECC Log Registers - ELOG0, ELOG1			
Bits	Name	Description	Default	Access	
31:2 4	(Reserved)		0	RO	
23:1 6		ECC Error Requester: Indicates the requester of the logged error. 0000_0000 - Intel XScale® Processor BIU 1000_0000 - IB BUS (North or South AHB) All Other Codes Are Reserved Note: Intel XScale processor BIU is logged for core transactions directed to the MCU via the Core MCU port only. Core transactions directed to the MCU via the IB port of the BIU will be logged as the IB BUS.	00H	RO	
15:1 3	(Reserved)		000 ₂	RO	
12		Read or Write: Indicates if the error occurred during a read or write transaction. 0 = Read error 1 = Write Error	0 ₂	RO	
11:0 9	(Reserved)		000 ₂	RO	
08		ECC Error Type: Indicates the type of error that occurred at this address. 0 = Single Bit Error 1 = Multi-Bit Error	0 ₂	RO	
07:0 0		Syndrome: Holds the syndrome value that indicated the error. Using the H-Matrix, this indicates the bit that caused the error if it was a single bit error.	00H	RO	



11.6.9 ECC Address Registers ECAR0, ECAR1

These registers are responsible for logging the addresses where the errors were detected on the local memory bus. Two errors can be detected and logged. The software knows which DDR1 SDRAM address had the error by reading these registers and decoding the syndrome in the log registers. For error details, see [“Error Correction and Detection” on page 614](#)).

Register Name:		ECC Address Registers - ECAR0, ECAR1																	
Hex Offset Address:		CC00 E528H CC00 E52CH				Reset Hex Value:				0x0000 0000H									
Register Description:		ECC Address Registers																	
Access: See below.																			
31																	02	01	00
																		(Rsvd)	

Register		ECC Address Registers - ECAR0, ECAR1		
Bits	Name	Description	Default	Access
31:0 2		Error Address: Stores the upper 30 bits of the address that resulted in a single bit or multi-bit error.	0	RO
01:0 0	(Reserved)		002	RO



11.6.10 ECC Test Register ECTST

This register allows testing between the ECC logic and the memory subsystem ([“ECC Testing” on page 625](#)). To test error handling software, the programmer writes this register with a non-zero masking function. Any subsequent writes to memory stores a masked version of the computed ECC. Therefore, any subsequent reads to these locations result in an ECC error.

Register Name:		ECC Test Register - ECTST																						
Hex Offset Address:		CC00 E530H				Reset Hex Value:		0x0000 0000H																
Register Description:		ECC Test Register																						
Access: See below.																								
31																08	07							0
(Reserved)																								

Register		ECC Test Register - ECTST		
Bits	Name	Description	Default	Access
31:0 8	(Reserved)		00 0000H	RO
07:0 0		ECC Mask: 8-bit ECC mask. Each bit of the generated ECC is XORed with the appropriate bit in this mask field before the ECC is stored into memory. See “ECC Testing” on page 625 .	00H	RW



11.6.11 Memory Controller Interrupt Status Register MCISR

Setting the MCISR asserts an interrupt to the core. Upon an interrupt, the Intel XScale processor polls the interrupt status register for each unit. The interrupt status register tells the core the reason for the interrupt. The MCU has three interrupt conditions: first ECC error (MCISR[0]), second ECC error (MCISR[1]), and more than two ECC errors (MCISR[2]).

If the MCU detects an ECC error and both MCISR[0] and MCISR[1] are cleared, the error is logged in ELOG0 and MCISR[0] is set to 1. If one of the MCISR bits are not clear and the MCU detects an error, the error is logged in the unused ELOGx register and the appropriate MCISR bit is set to 1. If both MCISR[0] and MCISR[1] are not clear, any additional ECC errors are not logged and MCISR[2] is set.

Bits 2:0 are read/clear bits which means that to clear them, software must write a one to these bits.

Register Name:	Memory Controller Interrupt Status Register - MCISR																						
Hex Offset Address:	CC00 E534H								Reset Hex Value:	0x0000 0000H													
Register Description:	Memory Controller Interrupt Status Register																						
Access: See below.																							
31																	05	04	03	02	01	00	
(Reserved)																							

Register		Memory Controller Interrupt Status Register - MCISR		
Bits	Name	Description	Default	Access
31:0 5	(Reserved)		0000 000H	RO
04		IB Discard Timer Expired: Indicates that the IB Port Delayed Read Completion Timeout has expired and a completion from the MCU has been discarded. 0 = No error detected 1 = Error detected	0 ₂	RW1C
03		Address Region Error: Indicates that a transaction to the invalid address region created with a 32-bit Region. (see Section 11.2.2.2, "DDR1 SDRAM Bank Sizes and Configurations" on page 590) 0 = No error detected 1 = Error detected	0 ₂	RW1C
02		ECC Error N: Indicates that the MCU detected an ECC error while MCISR[1] and MCISR[0] are both set. 0 = No error detected 1 = Error detected	0 ₂	RW1C
01		ECC Error 1: Indicates that the MCU detected an ECC error and recorded the error in ELOG1. 0 = No error detected 1 = Error detected and recorded in ELOG1	0 ₂	RW1C
00		ECC Error 0: Indicates that the MCU detected an ECC error and recorded the error in ELOG0. 0 = No error detected 1 = Error detected and recorded in ELOG0	0 ₂	RW1C



11.6.12 MCU Port Transaction Count Register MPTCR

Sets the number of transactions a given port can have processed during a single tenure. The 4-bit fields for each port allow up to 16 transactions to be processed by a port before the MCU arbiter selects a different port for DDRI SDRAM transactions. This register along with the “MCU Port Transaction Count Register MPTCR” on page 645 and “MCU Preemption Control Register MPCR” on page 645 used to optimize the memory controller operation.

Note: For the IXP45X/IXP46X network processors, this value MUST remain programmed to 11H in order to prevent unfair arbitration and indeterminate results.

Register Name:	MCU Port Transaction Count Register - MPTCR																						
Hex Offset Address:	CC00 E53CH					Reset Hex Value:			0x0000 0011H														
Register Description:	MCU Port Transaction Count Register																						
Access: See below.																							
31															08	07			04	03			00
(Reserved)																							

Register		MCU Port Transaction Count Register - MPTCR		
Bits	Name	Description	Default	Access
31:0 8	(Reserved)		000000H	RO
07:0 4		North and South AHB Transaction Count: Number of transactions the IB MCU port can have processed in a single tenure of the DDRI SDRAM. 1H = 1 transaction 2H = 2 transaction 3H = 3 transaction ... FH = 15 transactions 0H = 16 transactions	0001 ₂	RW
03:0 0		Core Transaction Count: Number of transactions the Core processor MCU port can have processed in a single tenure of the DDRI SDRAM. 1H = 1 transaction ... FH = 15 transactions 0H = 16 transactions	0001 ₂	RW

11.6.13 MCU Preemption Control Register MPCR

Enables Preemption of IB port transaction when a core processor transaction is pending.

The CMTQ must be the only memory transaction queue set to “High” (00₂) priority in the MACR, if preemption is to be enabled. If preemption is enabled when CMTQ priority is set to anything other than High, or additional ports are set to High, indeterminate results may occur. A special case is where there are only two ports, and the MACR is



not defined for programming. In this case the two ports are both set to High priority, but with only two agents at high, and no winner from the Medium priority level, the preemption control will work as expected.

Note: Preemption is not supported on the IXP45X/IXP46X network processors, therefore this register must remain set at its default value.

Register Name:	MCU Preemption Control Register - MPCR																		
Hex Offset Address:	CC00 E540H					Reset Hex Value:			0x0000 0000H										
Register Description:	MCU Preemption Control Register																		
Access: See below.																			
31															04	03			00
(Reserved)																	(Reserved)		

Register		MCU Preemption Control Register - MPCR		
Bits	Name	Description	Default	Access
31:0 4	(Reserved)		0	RO
03:0 0	(Reserved)	Preemption Data Phase Count: Specifies the number of DDRI data bursts that will complete before the current transactions will be preempted. The count is based from the beginning of the transaction. When a core transaction port request is detected after the count has been exceeded by the current transaction, the transaction will be preempted at the next burst length boundary. The current transaction must be a non-Core transaction. 0H = Disabled All other values are reserved.	0H	RW

11.6.14 Refresh Frequency Register RFR

The Refresh Frequency Register is programmed for refreshing the DD1R SDRAM subsystem at the specified interval. Writing to the RFR programs the refresh counter with the Refresh Interval. Reading from the RFR results in the value currently within the refresh counter. Refer to [“Typical Refresh Frequency Register Values” on page 614](#) for recommended programmed values.

Register Name:		Refresh Frequency Register - RFR																											
Hex Offset Address:		CC00 E548H						Reset Hex Value:		0x0000 0000H																			
Register Description:		Refresh Frequency Register																											
Access: See below.																													
31														12	11														00
(Reserved)																													



Register		Refresh Frequency Register - RFR		
Bits	Name	Description	Default	Access
31:1 2	(Reserved)		00000H	RO
11:0 0		Refresh Interval: Programs the number of clocks that triggers a request for a refresh cycle on the DDR1 SDRAM interface. If all zeroes, refresh cycles are disabled. See “Refresh Counter” on page 586 . Note: If the memory interface is busy when the refresh counter expires, it is possible for the MCU to generate more than one refresh cycle when the memory interface becomes available.	000H	RO

11.6.15 SDRAM Page Registers SDPR0-7

These registers hold the addresses of the eight pages that may be open. The valid bit is included. For more details, see [Section 11.2.2.6, “Page Hit/Miss Determination” on page 595](#). These read-only registers are provided for testability purposes. The page size is determined by the memory and data bus width as defined in [Table 203](#).

Register Name:		SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
Hex Offset Addresses:		0 — CC00 E550H 1 — CC00 E554H 2 — CC00 E558H 3 — CC00 E55CH 4 — CC00 E560H 5 — CC00 E564H 6 — CC00 E568H 7 — CC00 E56CH				Reset Hex Value:				0x0000 0000H																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
Register Description:		SDRAM Page Registers																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
Access: See below.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31																		12	11																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						

Register		SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7		
Bits	Name	Description	Default	Access
31:1 2		Page Address: Specifies the top 20 bits of an open page.	00000H	RW
11:0 1	(Reserved)		000H	RO
00		Page Valid: Indicates if the page address in bits 31 to 12 is a valid page. 0 = Page Invalid 1 = Page Valid	0 ₂	RW

§ §





12.0 Expansion Bus Controller

12.1 Overview

The Expansion bus controller provides an interface from the internal AHB in the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors to external Expansion target devices and an interface from external expansion master devices to the internal AHB. This interface can be used to couple/connect multiple IXP45X/IXP46X network processors to one another.

The Expansion bus controller includes a 25-bit address bus and a 32-bit wide data path. The expansion bus controller maps transfers between the internal AHB and external devices. The expansion bus supports Intel multiplexed, Intel non-multiplexed, Intel StrataFlash®, Synchronous Intel StrataFlash® Memory, Micron* Flow-Through ZBT, Motorola* multiplexed, Motorola non-multiplexed, and Texas Instruments* Host Port Interface (HPI) target devices. The Expansion bus controller also has an arbiter that supports up to four external devices that can master the Expansion bus. External masters can also access internal slaves such as the memory controller in the IXP45X/IXP46X network processors.

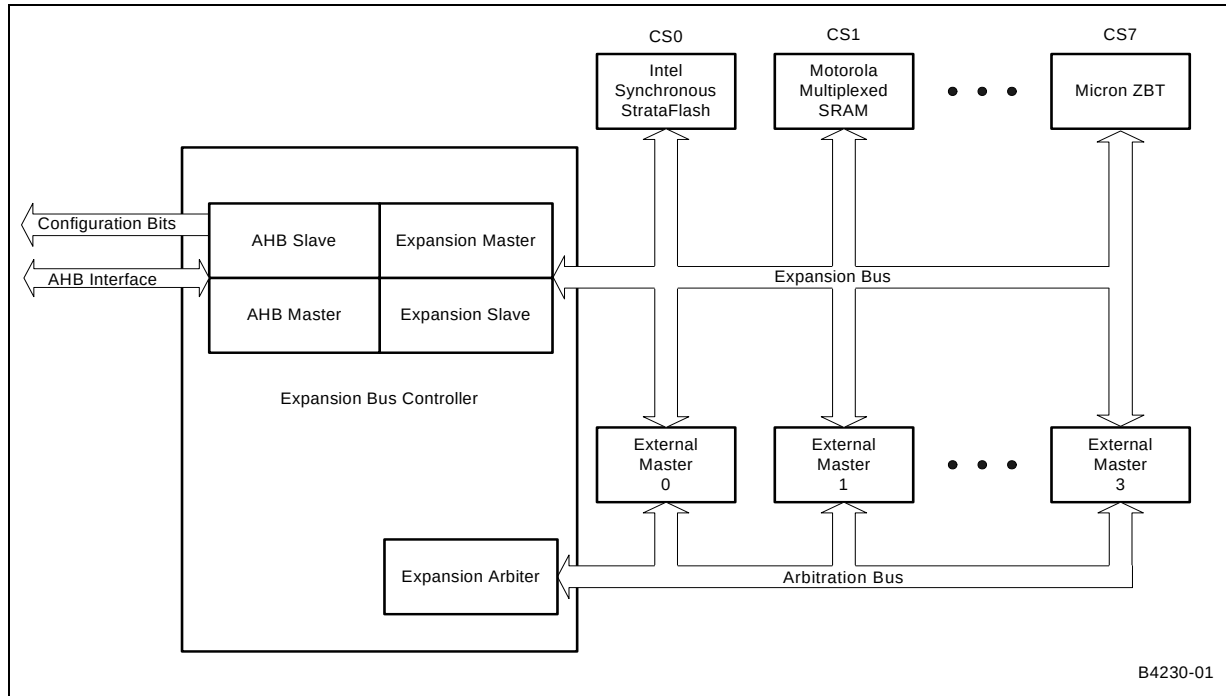
Applications having less than 32-bit external target devices may connect to an 8-bit or 16-bit (halfword) interface.

12.2 Feature List

- Outbound transfers (IXP45X/IXP46X network processors are the master to an external target device)
- Inbound transfers (IXP45X/IXP46X network processors are a target to an external master)
- Bus tri-state for sideband transfers (External masters accesses to external target device)
- Arbitration for four external masters with support for an external arbiter
- Eight programmable target chip selects
- Twenty five bits of address; thirty two bits of data
- Supports Intel mode and Motorola mode bus cycles
- Supports Intel StrataFlash®
- Supports 66-MHz Synchronous Intel StrataFlash® Memory (16-bit and 32-bit only)
- Supports 16-bit and 32-bit Micron Flow-Through ZBT (Zero bus turnaround) SRAMS
- Supports 8-bit and 16-bit Texas Instruments HPI specifications
- Multiplexed or non-multiplexed address / data busses for Intel/Motorola/HPI bus cycles
- Supports even and odd parity generation and calculation for Intel/Motorola/Micron ZBT modes
- Maximum clock input frequency of 80 MHz

12.3 Block Diagram

Figure 123. Expansion Bus Controller



12.4 Theory of Operation

The Expansion bus controller on the IXP45X/IXP46X network processors supports both of the following:

- outbound transfers that are initiated by an AHB master that targets Expansion bus targets
- inbound transfers that are initiated by an external Expansion bus master that targets internal IXP45X/IXP46X network processors slaves or external Expansion bus targets.

The Expansion bus controller arbiter has request/grant lines to support four external masters. An option to support an external arbiter is also supported. The Expansion data bus is 32 bits wide and the address bus is 25 bits wide.

Since the Expansion bus controller has only 1 outbound transaction queue, outbound accesses all complete in order. Similarly, all inbound transactions complete in order. The Expansion bus controller does not do any order checking between inbound and outbound transfer and relies on software for correct ordering.

12.4.1 Outbound Transfers

For outbound data transfers, the Expansion bus controller occupies 256 Mbytes of address space in the memory map of the IXP45X/IXP46X network processors and contains a 1-deep address queue, an 8-word write data fifo, and an 8-word read data FIFO. Eight chip selects are supported to allow up to eight independent external devices to be connected. The address space for each chip select is up to 32 Mbytes (16 Mbytes Intel® IXP400 Software-compatibility mode).



A clock input, EX_CLK, is required to operate the Expansion interface. The maximum clock frequency supported by the Expansion bus controller is 80 MHz. The clock input is provided to allow a wide variety of different peripherals to be connected to the Expansion interface. The GPIO unit provides a clock output after reset and may be used. Refer to the GPIO chapter for additional details. If a GPIO clock is used, it must be externally routed on the board to connect to EX_CLK. This implementation gives the designer the option to choose between a lower part count and the speed of the interface operations.

To provide a glue-less interface to a wide variety of devices, the Expansion bus controller supplies eight chip selects to a 32-bit wide external bus, which can be configured as Intel, Synchronous Intel, Micron ZBT, Motorola, or HPI-style controls. The signaling characteristics and timing for each chip select is individually programmable. After chip reset, chip-select 0 defaults to conservative timing values for controlling an asynchronous flash device and the data width of the flash is determined by the value of Expansion Bus Address bits 0 and 7 during the reset sequence. The remaining chip selects are un-programmed. Refer to ["Configuration Register 0" on page 706](#) for additional details.

For Synchronous Intel StrataFlash Memory, the Expansion bus controller only supports single word asynchronous page-mode read and synchronous burst-mode read (1-8 words). It does not support page mode read mode or single word latched asynchronous read mode. When configuring a Synchronous Intel StrataFlash Memory, wait polarity must be programmed to active low, data hold programmed to one clock, wait delay be deasserted with valid data and clock edge programmed to rising edge. For 16-bit Synchronous Intel devices, the burst length must be programmed to 16-word bursts. For 32-bit Synchronous Intel devices, the burst length must be programmed to 8-word bursts. The latency count must be programmed to the appropriate code that is defined in the Synchronous Intel StrataFlash Memory specification.

The Expansion Bus interface signals need to be connected based upon the device type (Intel, Synchronous Intel, Micron ZBT, Motorola, or HPI-style control signals) and a sample mapping of the pins are shown in [Table 217](#).

Table 217. Example Expansion Bus Pin Mappings to Target Devices

Pin	Intel StrataFlash® 28F128J3D	Synchronous Intel StrataFlash Embedded Memory (P30)	Motorola* MCM6946	Micron* ZBT MT55L128L36F 1	TI* HPI TMS320UC5409
EX_ALE	OPEN	ADV#	OPEN	ADV/LD#	OPEN
EX_ADDR[24:0]	A[23:0]	A[24:1]	A[18:0]	SA	HCSEL,HCNTL,H BIL
EX_BE_N[3:0]	OPEN	OPEN	OPEN	BW[d:a]#	OPEN
EX_CS_N[7:0]	CE#	CE#	EN	CE#	hcs
EX_DATA[31:0]	D[15:0]	D[15:0]	DQ[7:0]	DQ	HD[7:0]
EX_IOWAIT_N	OPEN	OPEN	OPEN	OPEN	OPEN
EX_PARITY[3:0]	OPEN	OPEN	OPEN	DQ	OPEN
EX_RD_N	OE#	OE#	G_N	OE#	hr_w_n
EX_RDY_N[3:0]	OPEN	OPEN	OPEN	OPEN	hrdy
EX_WR_N	WE#	WE#	W_N	R/W#	hds1_n
Note: The # symbol or an _N suffix in a signal name indicates that the signal is asserted low.					



The EX_IOWAIT_N signal is available to be shared by the devices attached to chip 0 through 7, when the chip selects are configured in Intel or Motorola mode of operation. The EX_IOWAIT_N signal allows an external device to hold off completion of the read or write phase of a transaction until the external device is ready to complete the transaction.

Similarly, EX_RDY_N[3:0] are provided for chip selects 7 through 4, respectively. The EX_RDY_N[3:0] signals are used to hold off data transfers when chip selects 7 through 4 are configured in HPI mode. For example when chip select 5 is configured in HPI mode of operation, chip select 5 will no longer respond to the EX_IOWAIT_N signal and will only respond to the EX_RDY_N[1]. All other chip selects will respond to the EX_IOWAIT_N signal. Chip selects 7 through 4 are the only chip selects that can be configured in HPI mode of operation.

As a slave the Expansion bus supports the following AHB commands shown in [Table 218](#). Commands that are unsupported result in an AHB Error response. For HPI devices, only half-word read and write transfers are supported in HPI-8 and HPI-16 modes.

Table 218. Supported AHB Commands

AHB Commands	Supported	Notes/Restrictions
Single byte read	Yes	Byte reads to 16-bit devices are only supported if BYTE_RD16 is set in EXP_TIMING_CS register
Single halfword read	Yes	AHB ADDR[0] = '0'
Single word read	Yes	AHB ADDR[1:0] = "00"
INCR-4 word read	Yes	AHB ADDR[4:0] <= (less than or equal to) "10000". Burst will not cross a 8-word boundary
INCR-8 word read	Yes	AHB ADDR[4:0] = "00000"
WRAP-8 word read	Yes	AHB ADDR[1:0] = "00"
Single byte write	Yes, except it is not supported for Synchronous Intel devices.	
Single halfword write	Yes	AHB ADDR[0] = '0'
Single word write	Yes, except it is not supported for 16-bit Synchronous Intel devices	AHB ADDR[1:0] = "00"
INCR-4 write	Yes, but only to 32-bit Micron ZBT devices	AHB ADDR[4:0] <= "10000". Burst will not cross a 8-word boundary
INCR-8 write	Yes, but only to 32-bit Micron ZBT devices	AHB ADDR[4:0] = "00000"
WRAP-8 write	No	
INCR read/write PCI typically generates INCR transfers	Yes, but only to 32-bit Micron ZBT devices	Burst will not cross a 8-word boundary. For incr reads, the Expansion bus controller reads data until the 8-word boundary is reached before returning the data to the AHB bus.
INCR-16 read/write	No	
WRAP-16 read/write	No	



12.4.1.1 Expansion Bus Address Space

Table 219. Trimmed Version of IXP45X/IXP46X network processors Memory Map

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 Mbyte	Expansion bus (Boot up)
0000_0000	3FFF_FFFF	1 Gbyte	DDR1 SDRAM
4000_0000	47FF_FFFF	128 Mbyte	(Reserved)
4800_0000	4FFF_FFFF	128 Mbyte	PCI
5000_0000	5FFF_FFFF	256 Mbyte	Expansion bus
6000_0000	63FF_FFFF	64 Mbyte	Queue manager

As seen in [Table 219](#), on the AHB, the lowest 256 Mbytes of address space (0x00000000 to 0x0FFFFFFF) is overlapped with the DDR1 SDRAM address space (0x00000000 to 0x3FFFFFFF). The actual interface that is accessed when the overlapped region is addressed is configurable based on the value of a configuration register bit located in the Expansion bus controller.

When bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1, the Expansion Bus accesses occupy the lowest 256-Mbytes of address space. When bit 31 of the Configuration Register 0 (EXP_CNFG0) is cleared to logic 0, the DDR1 SDRAM occupies the lowest 256-Mbytes of address.

On reset, bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1. This setting is required to allow the boot memory to be accessed which is located at hexadecimal address 0x00000000 in non-volatile storage on the Expansion Bus.

The first instruction execution of the Intel XScale processor is located at address 0x00000000. Once the boot sequence starts, the Intel XScale processor will switch bit 31 of the Configuration Register 0 (EXP_CNFG0) from logic 1 to logic 0, at an appropriate time.

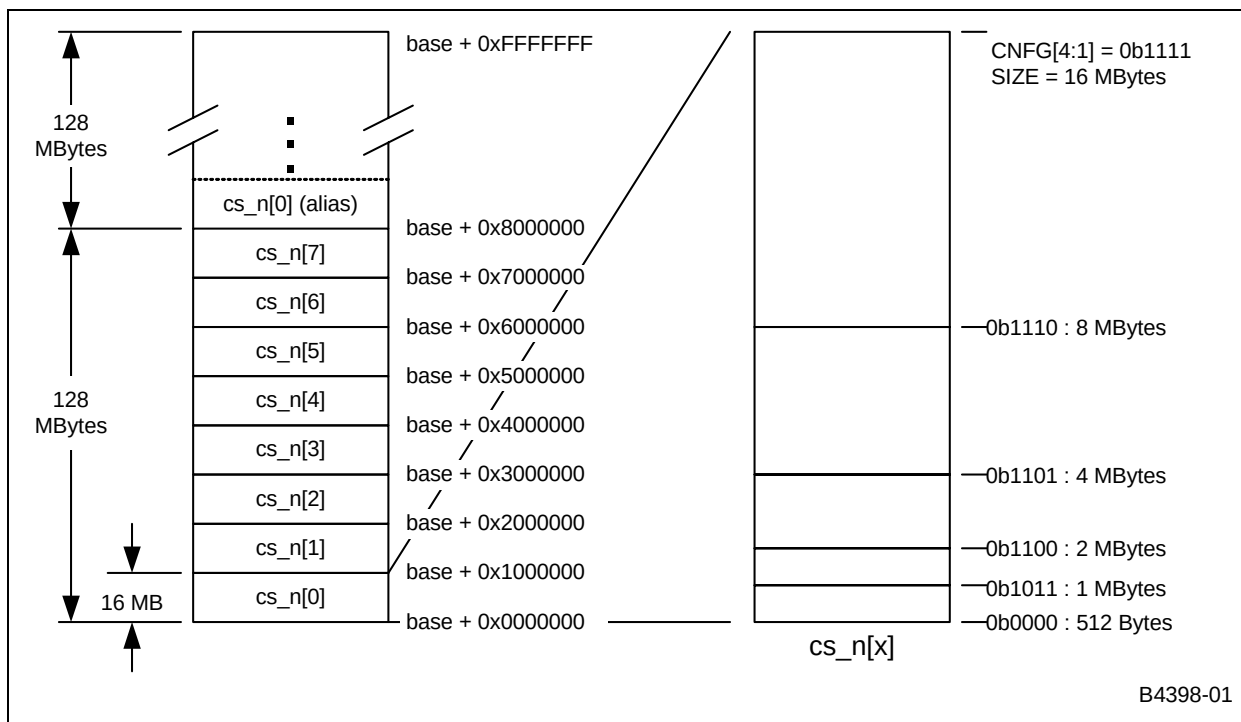
The information transfer from the flash to the DDR1 SDRAM can be completed this way:

The configuration bit can be swapped to allow the DDR1 SDRAM to have access at address 0x00000000 and the remainder of the flash information can be retrieved from the expansion bus address location 0x50000000 to 0x5FFFFFFF

12.4.1.2 Chip Select Address Allocation

The Expansion bus controller occupies 256 Mbytes of address space in the memory map of the IXP45X/IXP46X network processors. The Expansion bus controller uses bits 27:0, from the AHB, to determine how to translate the AHB address to the Expansion Bus Address. In order to maintain software compatibility with the IXP42X processors, the address mapping depends on if there are 32 Mbyte devices programmed in any of the EXP_TIMING_CS registers. Previous IXP42X processors did not support 32 Mbyte devices and each chip select was allocated 16 Mbyte of addressing. With the addition of support for 32 Mbyte devices, a new memory map is created so that each chip select is allocated 32 Mbyte of addressing. If there are no 32 Mbyte devices programmed (i.e All eight EXP_TIMING_CS registers bit 9 equal 0), the lower 24 bits of the AHB address are translated to the lower 24 bits of the Expansion Bus address, EX_ADDR [23:0]. EX_ADDR[24] will always be zero. Bits 26:24 of the AHB are used to decode one of eight chip-select regions implemented by the Expansion bus, each region being 16 Mbytes. Address bit 27 is not used and will currently alias each chip select region as shown on the left side of [Figure 124](#).

Figure 124. Chip Select Address Allocation when there are no 32-MByte Devices Programmed



The right side of Figure 124 shows the implementation of bit 13:10 of the each Timing and Control (EXP_TIMING_CS) Register. A Timing and Control (EXP_TIMING_CS) Register is implemented for each of the eight chip selects. Each chip select defines a base region size of 512 bytes with the actual size of the region given by the formula shown in Figure 125. If the AHB address is outside of the programmed region, the Expansion bus controller responds with an error response.

Figure 125. Expansion Bus Memory Sizing

$$\text{Region Size} = 2^{(9 + \text{CNFG}[4:1] + 16 * \text{CNFG}[0])}$$

For Examples of how to use this feature:

If bits 13:9 of Timing and Control (EXP_TIMING_CS0) Register 0 = "00000" an address space of $2^9 = 512$ Bytes is defined for chip select 0 EX_CS_N[0].

If bits 13:9 of Timing and Control (EXP_TIMING_CS1) Register 1 = "10000" an address space of $2^{17} = 128\text{KBytes}$ is defined for chip select 1 EX_CS_N[1].

If bits 13:9 of Timing and Control (EXP_TIMING_CS2) Register 2 = "11110" an address space of $2^{24} = 16\text{Mbytes}$ is defined for chip select 2 EX_CS_N[2].

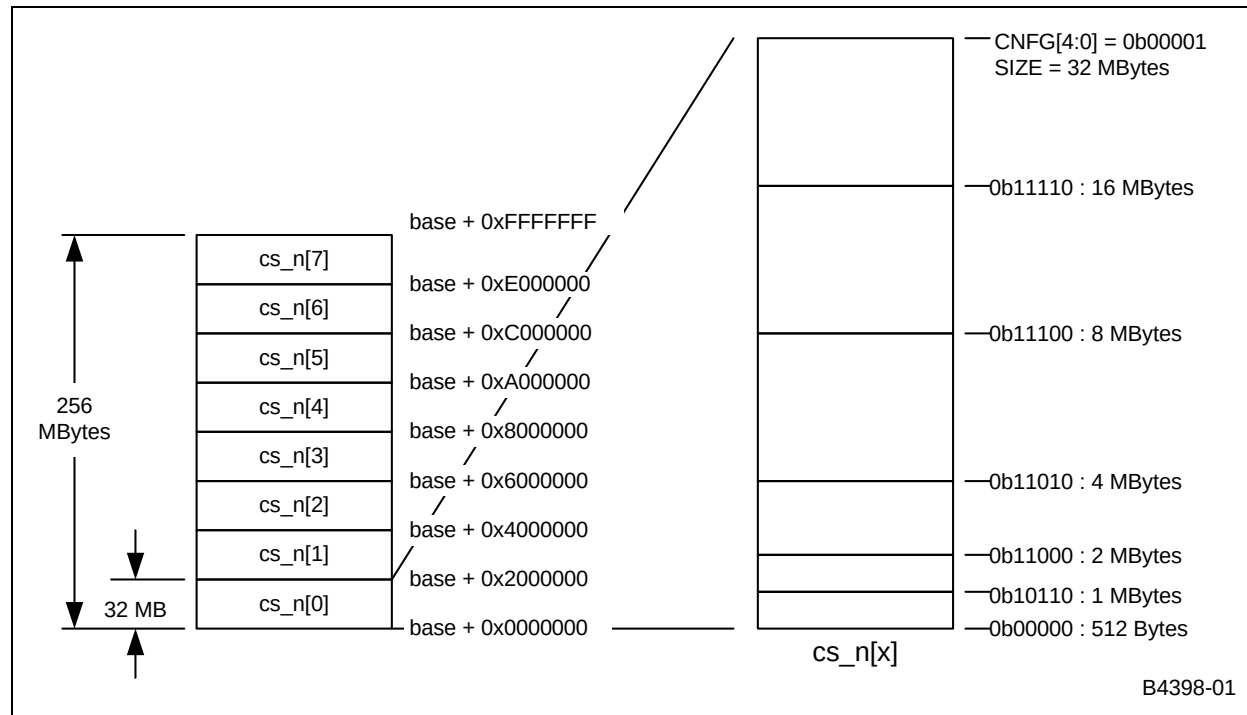
If bits 13:9 of Timing and Control (EXP_TIMING_CS7) Register 7 = "00001" an address space of $2^{25} = 32\text{Mbytes}$ is defined for chip select 7 EX_CS_N[7].

If there is a 32-MByte device programmed in any of the eight EXP_CS_TIMING registers, a different memory map is used as shown in Figure 126. The lower 25 bits of the AHB address are translated to the lower 25 bits of the Expansion Bus address, EX_ADDR [24:0]. Bits 27:25 of the AHB are used to decode one of eight chip-select regions implemented by the Expansion bus, each region being 32 Mbyte. If a design



has 16-MByte or smaller devices on all of the chip selects, one of the EXP_CS_TIMING register could be programmed to a 32-Mbyte device so the Expansion bus address mapping will not change if that design switches to 32-MByte device sometime in the future. The Expansion bus controller will still work with the smaller device, however an error response will not be generated if there is an access outside the device window for that device.

Figure 126. Chip Select Address Allocation when a 32-Mbyte Device is Programmed



12.4.1.3 Address and Data Byte Steering

Table 220 shows the address and data mapping from the AHB to the Expansion Bus. This table applies to Intel, Synchronous Intel, Micron ZBT and Motorola defined cycles only. For 32-bit read operations to a byte/halfword wide interface, multiple bytes are collected and then transferred as a complete 32-bit word. This pattern occurs as shown below for any allowable sub-length read access. Four- and eight-word reads are also supported and generate multiple accesses to the target device. Four- and eight-word reads to Synchronous Intel only generate one burst access to the device. Byte enables are generated for both reads and writes and are valid the same cycles (T1-T4 phases) as EX_ADDR is valid. Byte write devices (devices that need EX_BE_N asserted in the same exact cycles that EX_WR_N is asserted) are not supported. The AHB bus is always big-endian format and the Expansion bus is little-endian. The conversions for each cycle are shown in Table 220.

For 8-bit devices, EX_DATA[31:8] must not toggle to conserve power. Similarly, for 16-bit devices, EX_DATA[31:16] must not toggle. For sub-word writes to 32-bit devices, EX_DATA must not toggle for byte enables not asserted.



Table 220. Expansion Bus Address and Data Byte Steering (Sheet 1 of 3)

AHB Bus Cycle	Device Width Connected to Expansion Bus (8-bit or 16-bit or 32-bit)	AHB Address Value (AHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and AHB Data Bus
Word write	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
			01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
			10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
			11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Word write	16-bit	00	00	AHB data bus [31:16] = Expansion data bus [15:0], EX_BE_N = 0xC
			10	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Word write	32-bit	00	00	AHB data bus [31:0] = Expansion data bus [31:0], EX_BE_N = 0x0
Word read	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
			01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
			10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
			11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Word read	16-bit	00	00	AHB data bus [31:16] = Expansion data bus [15:0], EX_BE_N = 0xC
			10	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Word read	32-bit	00	00	AHB data bus [31:0] = Expansion data bus [31:0], EX_BE_N = 0x0
Halfword read	8-bit	0x	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
			01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
Halfword read	8-bit	1x	10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
			11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Halfword read	16-bit	0x	00	AHB data bus [31:16] = Expansion data bus [15:0], EX_BE_N = 0xC
Halfword read	16-bit	1x	10	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Halfword read	32-bit	0x	00	AHB data bus [31:16] = Expansion data bus [31:16], EX_BE_N = 0x3
Halfword read	32-bit	1x	00	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Halfword write	8-bit	0x	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
			01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE



Table 220. Expansion Bus Address and Data Byte Steering (Sheet 2 of 3)

AHB Bus Cycle	Device Width Connected to Expansion Bus (8-bit or 16-bit or 32-bit)	AHB Address Value (AHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and AHB Data Bus
Halfword write	8-bit	1x	10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
			11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Halfword write	16-bit	0x	00	AHB data bus [31:16] = Expansion data bus [15:0], EX_BE_N = 0xC
Halfword write	16-bit	1x	10	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Halfword write	32-bit	0x	00	AHB data bus [31:16] = Expansion data bus [31:16], EX_BE_N = 0x3
Halfword write	32-bit	1x	00	AHB data bus [15:0] = Expansion data bus [15:0], EX_BE_N = 0xC
Byte read	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	8-bit	01	01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	8-bit	10	10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	8-bit	11	11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	16-bit	00	00	AHB data bus [31:24] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte read	16-bit	01	00	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	16-bit	10	10	AHB data bus [15:8] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte read	16-bit	11	10	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte read	32-bit	00	00	AHB data bus [31:24] = Expansion data bus [31:24], EX_BE_N = 0x7
Byte read	32-bit	01	00	AHB data bus [23:16] = Expansion data bus [23:16], EX_BE_N = 0xB
Byte read	32-bit	10	00	AHB data bus [15:8] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte read	32-bit	11	00	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	8-bit	01	01	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	8-bit	10	10	AHB data bus [15:8] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	8-bit	11	11	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	16-bit	00	00	AHB data bus [31:24] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte write	16-bit	01	00	AHB data bus [23:16] = Expansion data bus [7:0], EX_BE_N = 0xE



Table 220. Expansion Bus Address and Data Byte Steering (Sheet 3 of 3)

AHB Bus Cycle	Device Width Connected to Expansion Bus (8-bit or 16-bit or 32-bit)	AHB Address Value (AHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and AHB Data Bus
Byte write	16-bit	10	10	AHB data bus [15:8] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte write	16-bit	11	10	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE
Byte write	32-bit	00	00	AHB data bus [31:24] = Expansion data bus [31:24], EX_BE_N = 0x7
Byte write	32-bit	01	00	AHB data bus [23:16] = Expansion data bus [23:16], EX_BE_N = 0xB
Byte write	32-bit	10	00	AHB data bus [15:8] = Expansion data bus [15:8], EX_BE_N = 0xD
Byte write	32-bit	11	00	AHB data bus [7:0] = Expansion data bus [7:0], EX_BE_N = 0xE

12.4.1.4 Expansion Bus Interface Configuration

There are eight registers called the Timing and Control (EXP_TIMING_CS) Registers that define the operating mode for each chip select. When designing with the Expansion Bus Interface, placing the devices on the correct chip selects is required.

Chip Select 0 through 7 can be configured to operate with devices that require an Intel, Synchronous Intel, Micron* ZBT or Motorola* Micro-Processor style bus accesses. These chip selects can be configured to operate in a multiplexed or a simplex mode of operation for either Intel- or Motorola-style bus accesses. Additionally, Chip Select 4 through 7 can be configured to generate Texas Instruments HPI-style bus accesses. The mode of operation (Intel, Motorola, TI* HPI*, or Synchronous Intel, Micron ZBT) is set by bits 15,14, and 8 of each Timing and Control (EXP_TIMING_CS) Register. [Table 228 on page 705](#) shows the possible settings for the Cycle Type selection using bits 15, 14, and 8 of the Timing and Control (EXP_TIMING_CS) Register.

Once the cycle type has been determined, the mode of operation must be set. There are two configurable modes of operation for each chip select, multiplexed and non-multiplexed. Bit 4 of the Timing and Control (EXP_TIMING_CS) Registers is used to select this mode. If bit 4 of the Timing and Control (EXP_TIMING_CS) Register is set to logic 1, the access mode for that Chip Select is multiplexed. Likewise, if bit 4 of the Timing and Control (EXP_TIMING_CS) Register is cleared to logic 0, the access mode for that Chip Select is non-multiplexed. For Synchronous Intel, Micron ZBT memories bit 4 must be programmed to logic 0. Multiplexed and non-multiplexed can imply different operations depending upon the Cycle Type that is selected. For more information refer to section [“Expansion Bus Outbound Timing Diagrams” on page 665](#).

The size of the data bus for each device connected to the Expansion bus must be configured. The data bus size is selected on a per-chip-select basis, allowing the most flexibility when connecting devices to the Expansion bus. There are three valid selections that can be configured for each data bus size, 8-bit, 16-bit, or 32 bit. Bit 0 and bit 2 of each Timing and Control (EXP_TIMING_CS) Register is used to select the data bus size on a per-chip-select basis. For more information refer to [Table 228, “Bit Level Definition for each of the Timing and Control Registers” on page 705](#). One special case for the data bus width selection is for chip select 0. Chip select 0 (data bus width) is selected by the value contained on Expansion Bus Address bit 0 and bit 7 at the assertion of PLL_LOCK. When PLL_LOCK is deasserted, Expansion Bus Address bit 0 and 7 will be captured into Timing and Control (EXP_TIMING_CS) Register bit 0 and 2 for Chip Select 0. This feature allows either an 8-bit, 16-bit or 32-bit flash device to be connected to the Expansion Bus Interface for a boot device.



Each chip select can be independently enabled or disabled by setting a value in bit 31 of each Timing and Control (EXP_TIMING_CS) Register. Clearing bit 31 of the Timing and Control (EXP_TIMING_CS) Register to logic 0 disables the corresponding chip select. Setting bit 31 of the Timing and Control (EXP_TIMING_CS) Register to logic 1 enables the corresponding chip select. Accesses to chip selects that are disabled result in an AHB error response.

Split transfers are supported for all read transfer types and controlled by setting bit 3 (SPLT_EN) of the Timing and Control (EXP_TIMING_CS) Register. Setting bit 3 of each Timing and Control (EXP_TIMING_CS) Register to logic 1 enables split transfers for accesses to the corresponding chip select. Clearing bit 3 of each Timing and Control (EXP_TIMING_CS) Register to logic 0 disables split transfers for accesses to the corresponding chip select. Multi-word read transfers requested by the AHB might be split. Only one access at a time may be split.

Split transfers require that the read data from the Expansion bus be stored in an eight-word FIFO until all expansion bus transfers are complete before that data is forwarded on the AHB. When split transfers are initiated, the Expansion bus controller acknowledges the read request. The AHB will be relinquished until all the data is acquired from the Expansion bus and stored in the eight-word FIFO contained in the Expansion bus controller. After all of the data has been acquired by the Expansion bus controller, the requesting master on the AHB will be signaled that the read data is in the FIFO and the read transfer will complete — uninterrupted in its normal rotation in the arbitration scheme. This feature allows for slow devices connected to the Expansion bus not to impede the performance of data flow from high-speed peripherals (like PCI) on the AHB.

Retries also are supported and used predominately when expansion bus requests are issued while a split transfer is in progress. Retries may also occur when a write occurs when a previous read or write is in progress. A split response will never occur during a write transfer. The Expansion bus controller will never deassert AHB HREADY for writes; the writes will be posted or retried. The Expansion bus controller will also retry reads until all the words of the Expansion bus data transfer are in the data FIFO if SPLT_EN is clear. The Expansion bus controller will not deassert AHB HREADY for reads.

Each chip select region has the ability to be write-protected by setting bit 1 of each Timing and Control (EXP_TIMING_CS) Register. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is cleared to logic 0, writes to a specified chip select region results in an error response. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is set to logic 1, writes are allowed to a specified chip select region. Chip select 0 will be write-protected after reset.

For chip selects 4 through 7 configured in HPI mode of operation, there is an associated ready bit EX_RDY_N[3:0]. The ready bit is only used when the mode of operation is set to Texas Instruments HPI mode. The ready bits are used to hold off the Intel XScale processor when the given DSP is not ready to complete the transfer.

However, the polarity of this ready bit can vary based upon the DSP that is selected. Bit 5 of each Timing and Control (EXP_TIMING_CS) Register allows the polarity used by each ready bit to be independently set. When bit 5 of the Timing and Control (EXP_TIMING_CS) Register is cleared to logic 0, the ready bit is cleared to respond to an active low signal (logic 0). When bit 5 of the Timing and Control (EXP_TIMING_CS) Register is set to logic 1, the ready bit is set to respond to an active high signal (logic 1).

One final set of parameters that may be set prior to using Expansion Bus Interface Chip Select 1 through Chip Select 8. After boot up, these parameters may be adjusted for Chip Select 0 as well. These five parameters are the timing extension parameters for each phase of an Expansion Bus access.

There are five phases to every Expansion Bus access:



- T1 – Address Timing
- T2 – Setup/Chip Select Timing
- T3 – Strobe Timing
- T4 – Hold Timing
- T5 – Recovery Phase

For Synchronous Intel mode, the T1, T2, T3, T4, T5 timing parameters are only used for writes. For Synchronous Intel reads, the Expansion bus controller uses the Count value programmed in the EXP_SYNCINTEL_COUNT register to determine how many cycles before data is valid. For Micron ZBT devices, the timing parameters must be programmed to all zero since reads and writes are synchronous.

The expansion bus address is used to present the 25 bits of the address [24:0] used for the Expansion bus access accompanied by an address latch enable output signal, EX_ALE for multiplexed devices. The address phase normally lasts two clock cycles in multiplexed mode. The address phase may be extended by one to three clock cycles using the T1 - Address Timing parameter, bits 29:28 in the Timing and Control (EXP_TIMING_CS) Register for the particular Chip Select. When the address phase T1 is extended, the ALE pulse is extended and always deasserts one cycle prior to the end of the T1 phase. The lower address bits are placed onto the data bus (i.e for a 16 bit data bus, EX_DATA contains EX_ADDR[15:0]) along with EX_ADDR[24:0] signals during the first cycle of the address phase. During the second cycle of the address phase, the data bus now will output data when attempting to complete a write or tri-state when attempting to complete a read. The address signals will retain their state.

For Synchronous Intel and Micron ZBT devices, EX_ALE acts as the address valid signal (ADV#) and is logic 0 during the address phase and logic 1 during the continuation of a burst or IDLE cycle.

Due to the fact that, in HPI mode of operation, it is possible to begin an access to a busy device (EX_RDY_N is false), special consideration must be taken with programming the T1 — Address Timing parameter when using the chip select in HPI mode. The T1 – Address Timing parameter must be set to a minimum of two additional cycles (T1 must equal to 0x2). Programming the T1 – Address Timing parameter to this value ensures that the asynchronous EX_RDY_N input is sampled and available to the controlling hardware logic before beginning the new HPI access over the Expansion bus.

The chip-select signal is presented for one Expansion bus phase before the Strobe Phase. The chip select will be presented for the remainder of the Expansion bus cycles (setup, strobe, and hold phases).

The Setup/Chip Select Timing phase may also be extended by one to three clock cycles, using bits 27:26 of the Timing and Control (EXP_TIMING_CS) Register, T2 – Setup/Chip Select Timing parameter. In HPI mode of operation, T2 is defined as the time required by the external DSP device to drive EX_RDY_N false for the current access plus the time required by the Expansion bus controller to sample and synchronize the EX_RDY_N signal. The T2 – Setup/Chip Select Timing parameter must have a minimum value of two additional cycles (T2 >= 0x2). Programming the T2 – Setup/Chip Select Timing parameter to be three clock cycles in length ensures that when the Strobe Phase, T3, begins, the Strobe Phase will be able to sample the EX_RDY_N signal and exit the Strobe Phase at the proper time.

The Strobe Phase of an expansion bus access is when the read or write strobe is applied. The 25 Expansion Bus Interface Address bits are maintained in non-multiplexed mode or the Expansion Bus Interface Data bus is switched from address to data when configured in multiplexed mode during the Strobe Phase.



The Strobe Phase may be extended from one to 15 clock cycles, as defined by programming bits 25:22 of the Timing and Control (EXP_TIMING_CS) Register, T3 – Strobe Timing parameter. In HPI mode of operation, the T3 – Strobe Timing parameter must have a minimum value of one additional cycle ($T3 \geq 0x1$). Programming the T3 – Strobe Timing parameter to be two clock cycles in length ensures that any data sent to the DSP is captured regardless of when the EX_RDY_N signal is asserted by the DSP.

The Hold Phase of an expansion bus access is provided to allow a hold time for data to remain valid after the data strobe has transitioned to an invalid state. During a write access, the Hold Phase provides hold time for data written to an external device on the Expansion bus, after the strobe pulse has completed.

During a read access, the Hold Phase allows an external device time to release the bus after driving data back to the controller. The Hold Phase may be extended one to three clock cycles, using bits 21:20 of the Timing and Control (EXP_TIMING_CS) Register, T4 – Hold Timing parameter. In HPI mode of operation, the Hold Phase is defined the same as described for the Intel and Motorola modes of operation, but must be set to a minimum value of one additional cycle ($T4 \geq 0x1$).

After the address and chip select is de-asserted, the Expansion bus controller can be programmed to wait a number of clocks before starting the next Expansion Bus access. This action is referred to as the Recovery Phase. The Recovery Phase may be extended one to 15 clock cycles using bits 19:16 of the Timing and Control (EXP_TIMING_CS) Register, T5 – Recovery Timing parameter. In HPI mode of operation, the Recovery Phase is defined the same as described for the Intel and Motorola modes of operation.

12.4.1.5 Using I/O Wait

The EX_IOWAIT_N signal is available to be shared by devices attached to chip selects 0 through 7, when configured in Intel or Motorola modes of operation. The main purpose of this signal is to properly communicate with slower devices requiring more time to respond during data access. During idle cycles, the board is responsible for ensuring that EX_IOWAIT_N is pulled-up. The Expansion bus controller will always ignore EX_IOWAIT_N for synchronous Intel mode writes.

As shown in [Figure 127](#), a normal phase transaction is initiated during the T1 (Address Timing) period, in which the processor drives the address lines with an address that is decoded by the peripheral being accessed.

The next segment of the transaction is the T2 (Chip Select Timing) period, in which the processor asserts Chip Select and the Address signals have reached a stable state. EX_IOWAIT_N must be asserted during the T2 period. If not asserted at this time, the processor ignores EX_IOWAIT_N and treats it as it never occurred. If EX_IOWAIT_N is asserted during T2, the processor expects the signal to be deasserted during the T3 (Strobe Timing) period.

T3 can be programmed from 0 to F, where the value indicates the number of cycles that the processor waits for EX_IOWAIT_N to be deasserted. The counter starts at the rising edge of the clock when EX_RD_N or EX_WR_N is asserted. The following rules describe processor operation during T3:

- If EX_IOWAIT_N is deasserted at least two clock cycles before the T3 counter expires, then the processor deasserts EX_RD_N/EX_WR_N at the end of the number of cycles programmed in T3.
- If EX_IOWAIT_N is deasserted after T3 counter has expired, it will take 2 more clock cycles before the processor deasserts the EX_RD_N/EX_WR_N signal.
- If EX_IOWAIT_N is not deasserted before the T3 counter expires, then EX_RD_N/EX_WR_N will continue to be asserted for as long as EX_IOWAIT_N continues to be asserted, plus 2 more clock cycles.



The T4 (Hold Timing) period is the time interval in which Chip Select will be held after READ is deasserted. T4 prevents bus contention while Chip Select is asserted, in case the peripheral driving the bus continues to send data out after READ has been deasserted. During T4 no other transaction can start, since the current transaction will not finish until Chip Select is deasserted by the processor.

T5 is the recovery time required before the next transaction can start.

The EX_IOWAIT_N signal only affects the interface during T2 when it is asserted and during T3 when it is deasserted. If Chip Selects 4 through 7 are configured in HPI mode of operation, each chip select will have a corresponding HRDY signal called EX_RDY. The polarity of the ready signal is programmable. Chip Select 4 corresponds to EX_RDY signal 0 and Chip Select 7 corresponds to EX_RDY signal 3.

In the case of extended phase timing, EX_IOWAIT_N is used in the same way as the normal phase, however, the T1, T2, T4 and T5 periods take place over 4 cycles. T3 is still programmable but each value is a multiple of 4 cycles. See [Figure 128](#) for details.

Figure 127. I/O Wait Normal Phase Timing

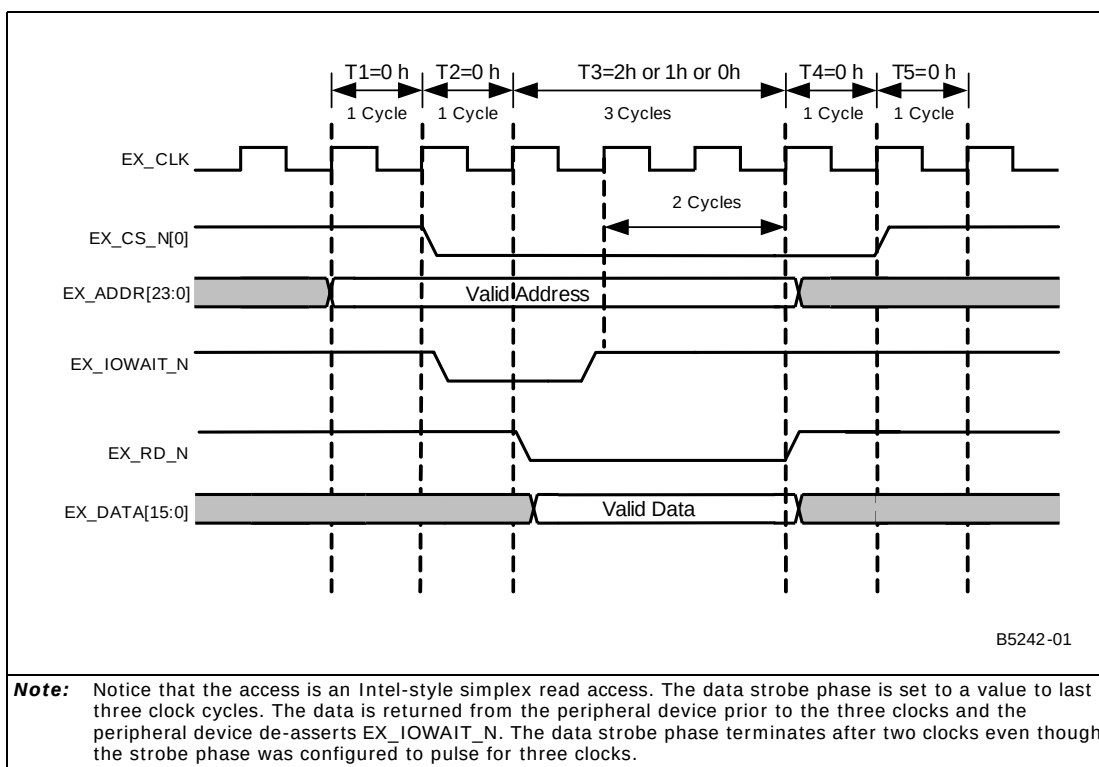
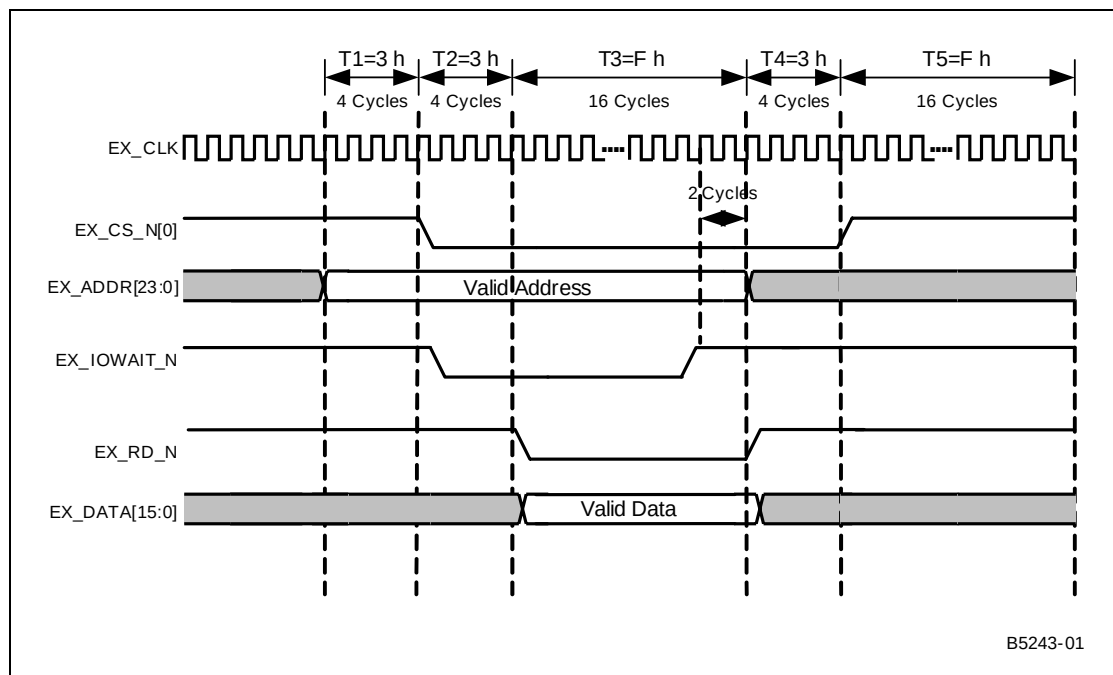




Figure 128. I/O Wait Extended Phase Timing



12.4.1.6 Parity

The Expansion bus controller generates even or odd parity for each byte written to EX_DATA and compares parity for each byte read on EX_DATA, if PAR_EN is set in each EXP_TIMING_CS register. EX_PARITY is transferred in the same clock cycle in which EX_DATA is transferred. If a read transfer results in a parity mismatch on EX_PARITY, the AHB address is logged in the EXP_PARITY_STATUS and OutErrorSts is set. The Expansion bus controller will then respond with an AHB error response on the first data cycle, even if the parity error does not occur on the first data cycle on the Expansion bus. Exp_parity_error will be asserted by the Expansion bus controller during a parity error and an interrupt will be generated if enabled in the interrupt controller. Exp_parity_error will remain asserted until software clears EXP_PARITY_STATUS register.

Even parity is defined as the number of 1's on EX_DATA[7:0] and EX_PARITY[0] must be an even number. For example, if EX_DATA[7:0] = 0x25, EX_PARITY[0] must be 1, since there are 3 bits set on 0x25 and there needs to be an even number of 1s. Parity for the second byte of EX_DATA is generated on EX_PARITY[1]; the third byte of EX_DATA is generated on EX_PARITY[2], and the 4th byte of EX_DATA is generated on EX_PARITY[3]. For 16-bit devices, EX_DATA[31:16] is not used and EX_PARITY[3:2] should not toggle and is ignored for reads. For 8-bit devices, EX_DATA[31:8] is not used and EX_PARITY[3:1] should not toggle and is ignored for reads. For 32-bit devices, byte and halfword writes should not toggle EX_PARITY on byte enables that are not asserted. For 32-bit devices, byte and half-word reads must not compare parity on bytes that are not read. If PAR_EN is cleared for a particular device, the Expansion bus controller doesn't generate or compare parity and EX_PARITY should not toggle to conserve power. Odd parity can be enabled by setting OddPar in the EXP_MST_CONTROL register and is equivalent to the inverted value of even parity.



12.4.1.7 Special Design Knowledge for Using HPI mode

The Expansion bus controller supports a number of the 8-bit and 16-bit versions of the Texas Instruments Host Port Interface* (HPI) standards. This flexibility allows the TMS320C54xx family of Digital Signals Processors (DSP) to seamlessly interface to the Expansion Bus for the IXP45X/IXP46X network processors.

If the Expansion Bus CS (Chip-Select) is configured to operate to operate in HPI-8 Mode, then a STRH (16-bit write) Intel XScale processor instruction must be used for writing to the HPI-8 device, even though it is in an 8-bit device. If a STRB (8-bit write) instruction is used instead, then the Intel XScale processor's data abort handler will be initiated, causing the assigned HPI-8 CS signal to deassert.

However, there are some special things to note when using the Expansion Bus in HPI mode of operation. These features are shown in the following tables. There are also some restrictions on the timing parameters and these are outlined in [Section 12.4.1.4, "Expansion Bus Interface Configuration"](#) on page 658.

The expansion bus address-pins bits 0, 1, 2, 22, and 23 are multiplexed with special function signal pins for HPI as shown in [Table 221](#).

Table 221. Multiplexed Output Pins for HPI Operation

HPI Control Signal	Output Signal Pin
EX_HBIL	EX_ADDR [0]
EX_HCNTL [1:0]	EX_ADDR [2:1]
EX_HCSEL [1:0]	EX_ADDR [23:22]

The byte identification signal, EX_HBIL, is used to determine the byte transfer order. (EX_HBIL is driven low for the first byte of the transfer and driven high for the second byte.)

The byte order bit (BOB) in the HPIC register (contained in the DSP) — within the HPI device — is used to determine the placement for the two bytes of the transfer. Please consult the datasheet of the specific DSP being connected to determine the order of the transferred bytes.

When operating in HPI mode, bits 13:10 in the Timing and Control (EXP_TIMING_CS) Registers are ignored.

When operating in HPI-16, non-multiplexed mode, the Expansion bus address bus provides direct accesses to the DSP memory space. The data associated with this address will be read or written from the location specified by the value contained on the Expansion Bus address bits.

The signals EX_HCNTL [1:0] are multiplexed onto the EX_ADDR [2:1] pins. When communicating to a multiplexed HPI interface, the EX_HCNTL [1:0] signals are used to select one of four internal registers used for interfacing to the DSP. The EX_HCNTL [1:0] mapping is described in the [Table 222](#).



Table 222. HPI HCNTL Control Signal Decoding

hcntl[1:0]	Required Access
00	Read / write control register (HPIC)
01	Read / write data register (HPID) HPI-8: Post-increment HPIA on reads, pre-increment on writes. HPI-16: Post-increment HPIA on reads and writes
10	Read / write address register (HPIA)
11	Read / write data register (HPID)

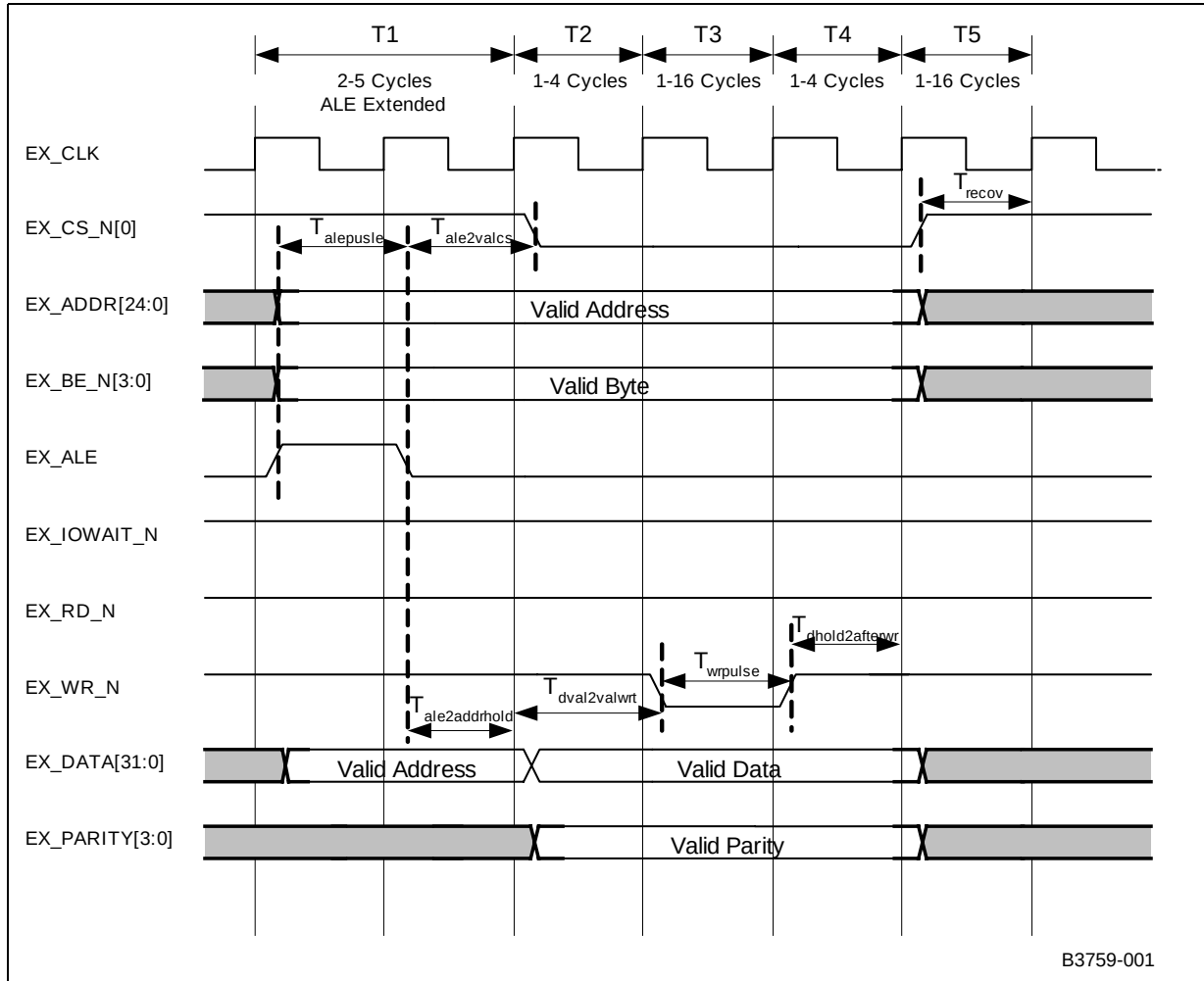
12.4.1.8 Expansion Bus Outbound Timing Diagrams

The STATE signal that is shown in some of the following timing diagrams is the internal state of the Expansion bus controller.



12.4.1.8.1 Intel, Multiplexed-Mode Write Access

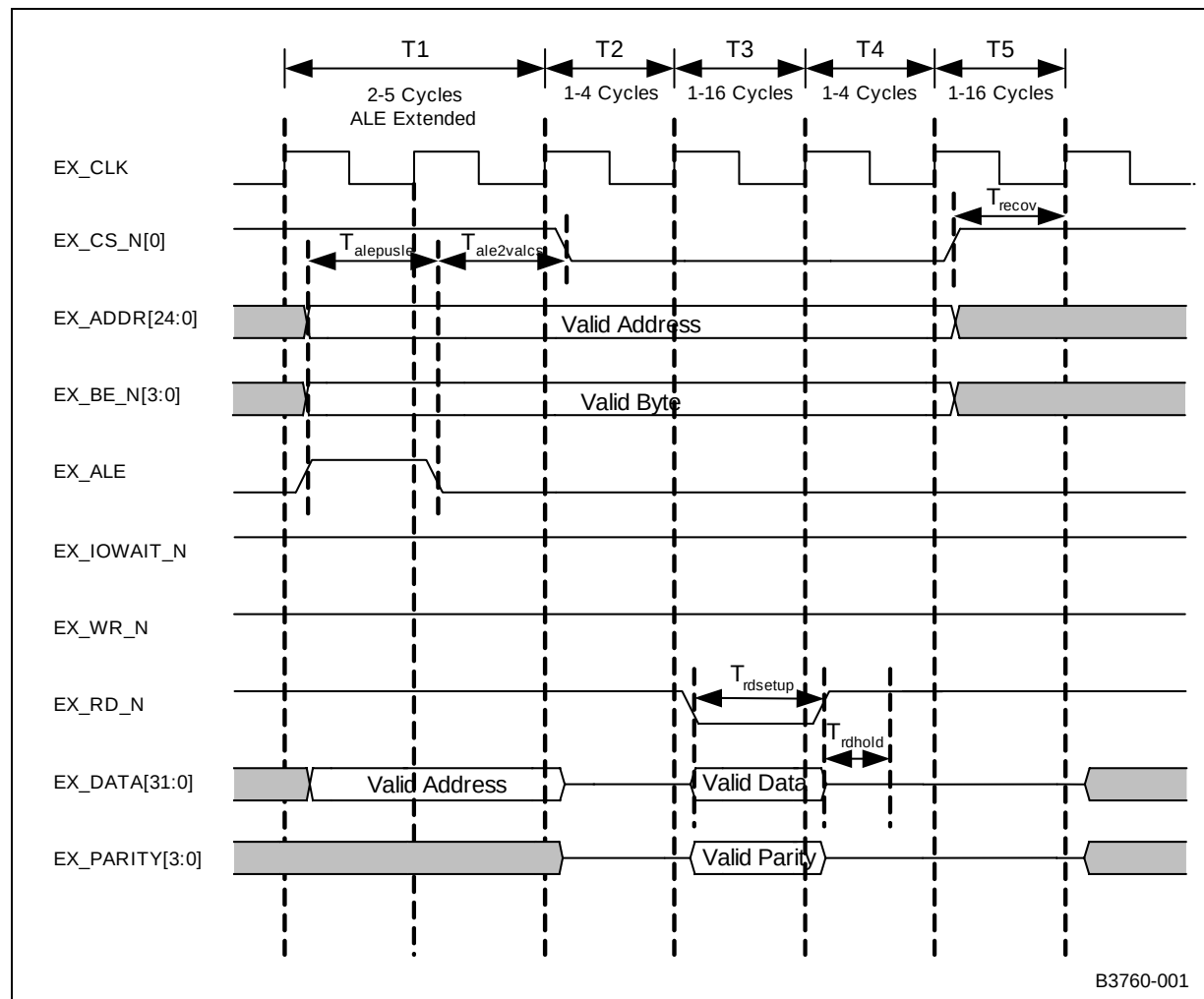
Figure 129. Expansion Bus Write (Intel, Multiplexed Mode)





12.4.1.8.2 Intel, Multiplexed-Mode Read Access

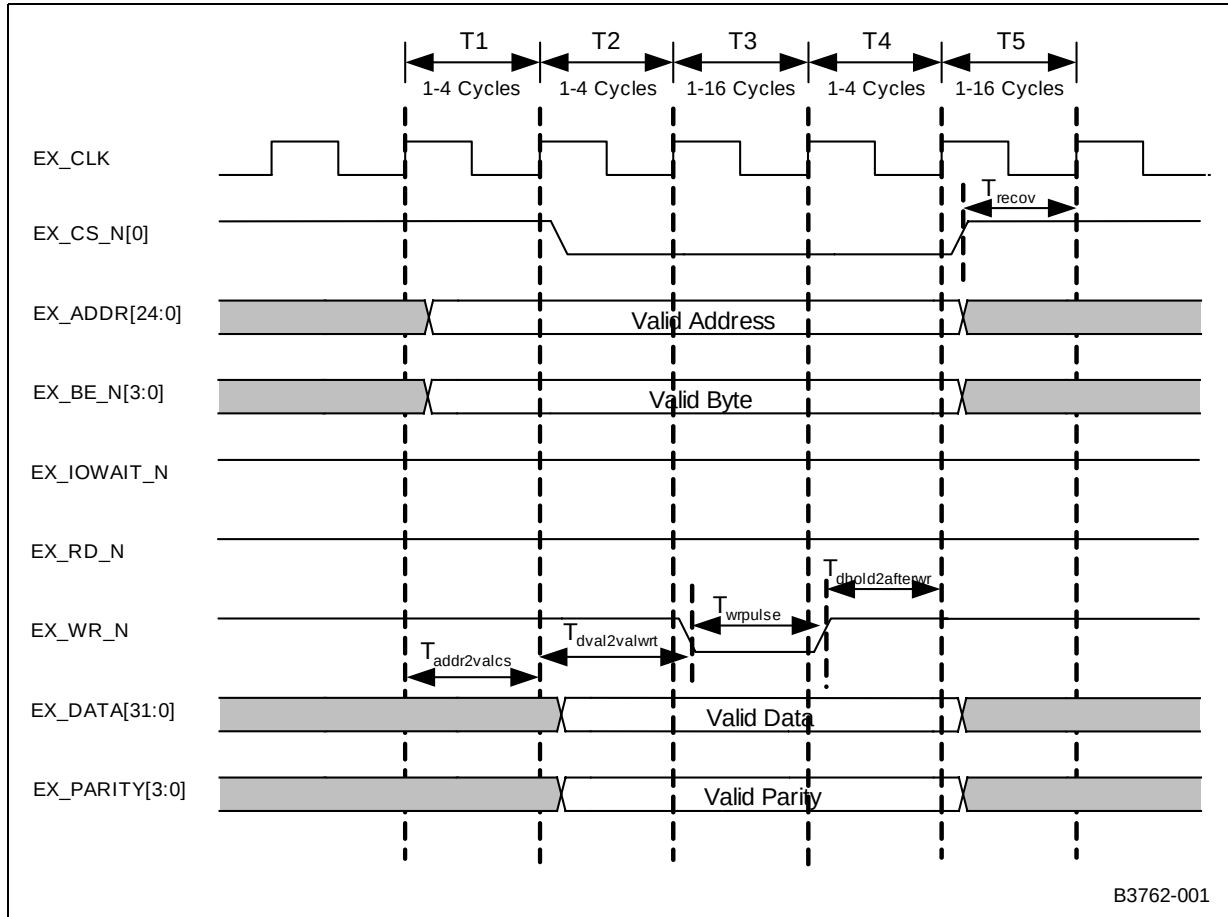
Figure 130. Expansion Bus Read (Intel, Multiplexed Mode)





12.4.1.8.3 Intel Simplex-Mode and Synchronous Intel Write Access

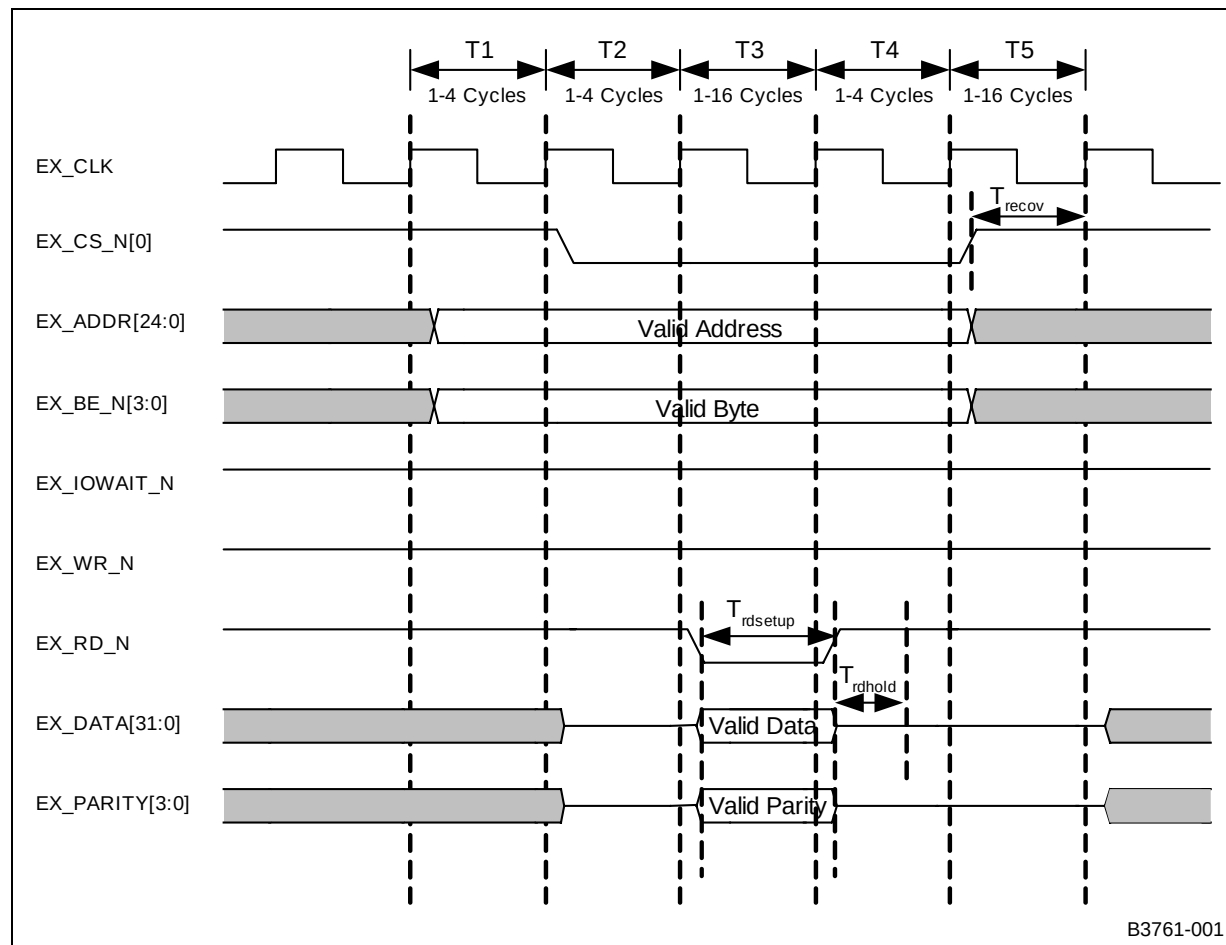
Figure 131. Expansion Bus Write (Intel Simplex-Mode, Synchronous Intel)





12.4.1.8.4 Intel Simplex-Mode Read Access

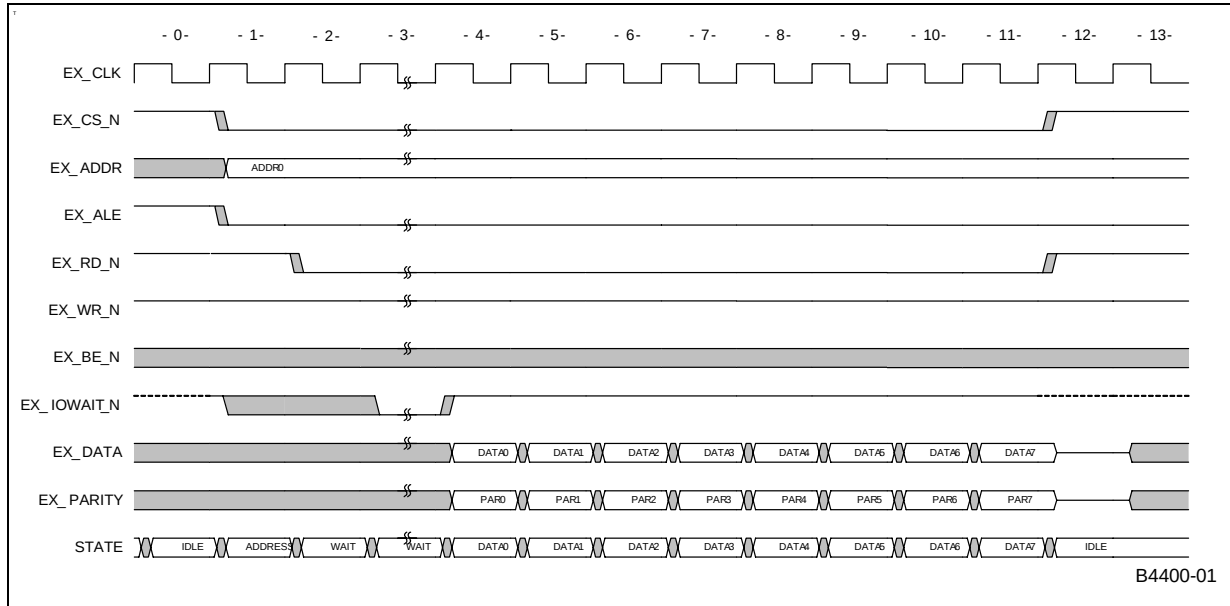
Figure 132. Expansion Bus Read (Intel Simplex-Mode)





12.4.1.8.5 Synchronous Intel 8-Word Read Access

Figure 133. Intel Synchronous 8-Word Read

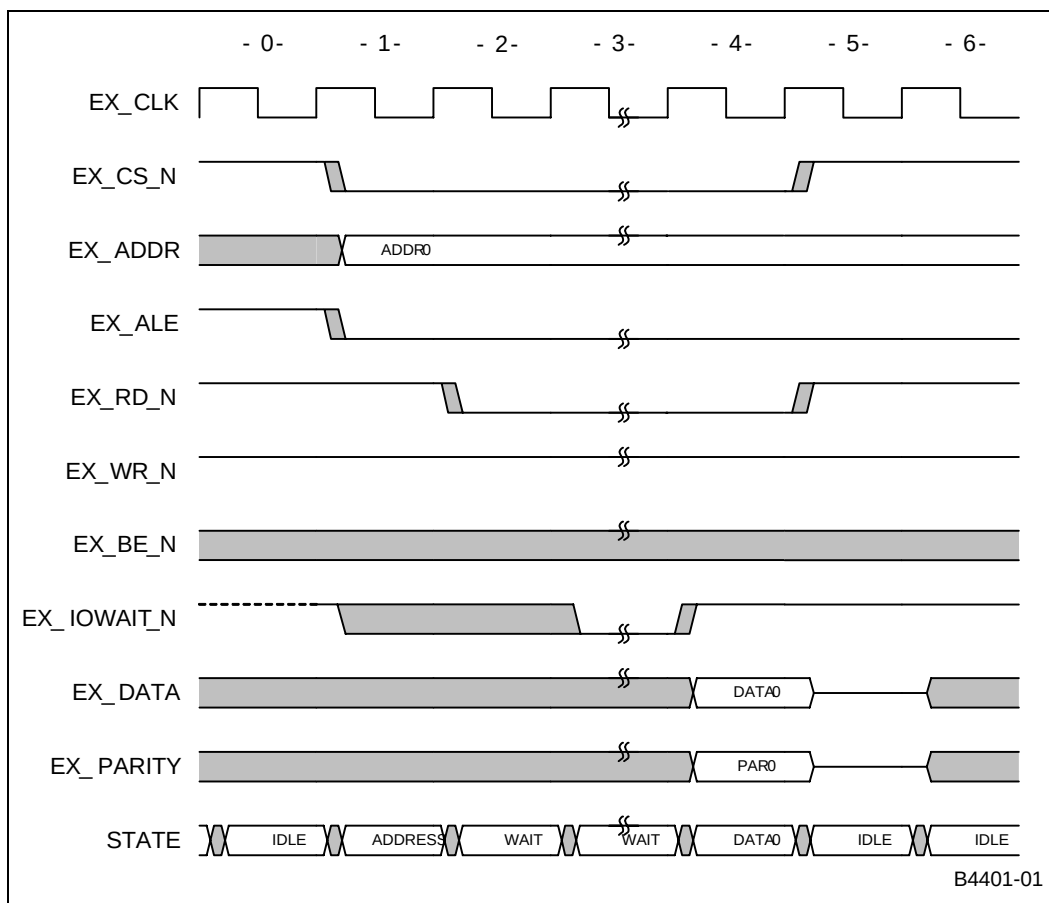


The above timing diagram shows an 8-word read to a Synchronous Intel device such as Synchronous Intel StrataFlash. Depending on the EX_CLK period, the latency count bits in the Intel Synchronous Device read configuration register needs to be programmed appropriately. The Expansion bus controller will always wait in cycles 1 and 2, regardless of EX_IOWAIT_N. The device will then assert EX_IOWAIT_N for several cycles and deassert EX_IOWAIT_N when its ready to transfer data. After the device deasserts EX_IOWAIT_N, it will transfer the remaining words until all 8 words are transferred. The Expansion bus controller and Synchronous Intel device both support wrapping for 8-word reads, therefore ADDR0 is not always aligned to an 8-word boundary. The STATE signal shows the internal Expansion bus state.



12.4.1.8.6 Synchronous Intel 1-Word Read Access

Figure 134. Intel Synchronous One-Word Read





12.4.1.8.7 Micron* ZBT Write / Read / Write Access

Figure 135. Micron* ZBT Write / Read / Write

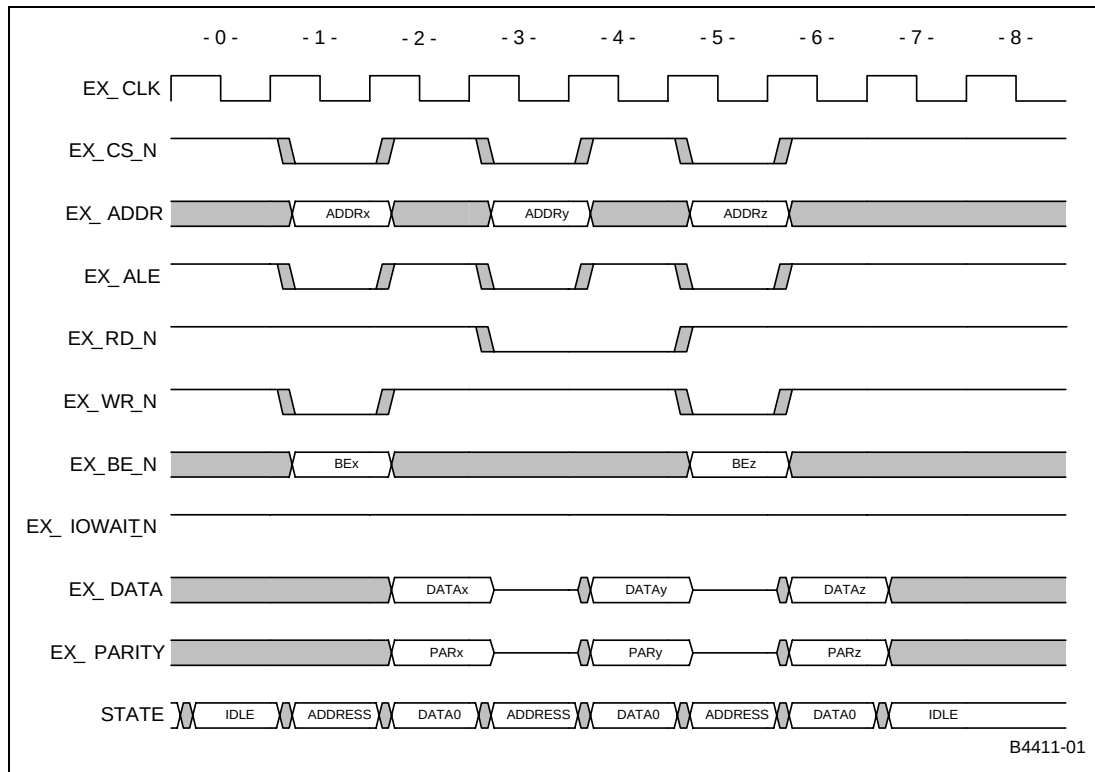
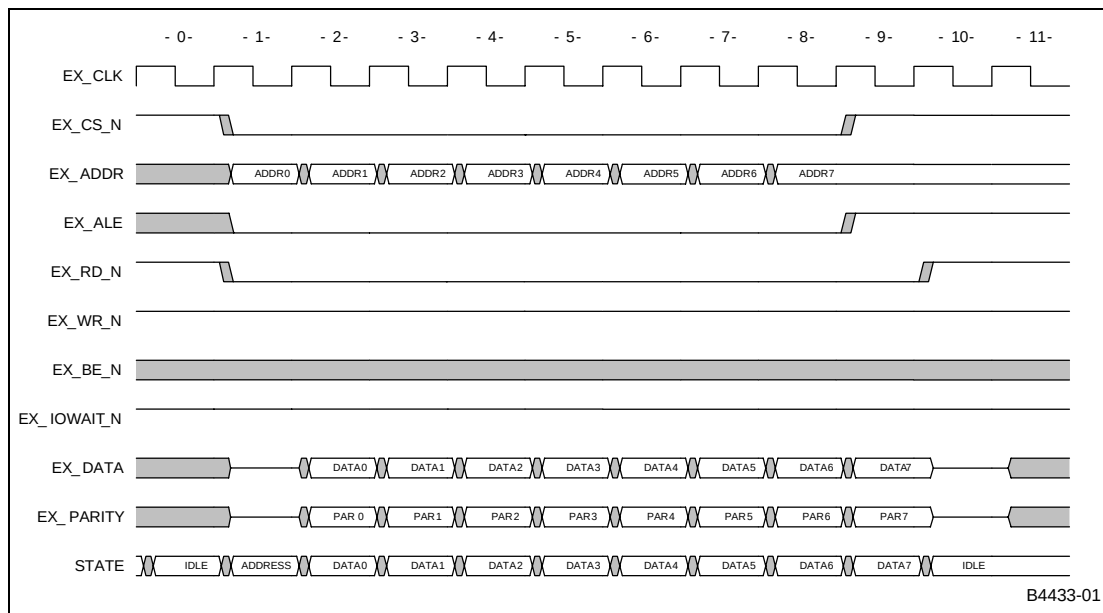


Figure 135 shows a write to a Micron ZBT device followed by a read and then followed by another read. The device will never assert EX_IOWAIT_N for this mode.



12.4.1.8.8 Micron* ZBT 8-Word Read Access

Figure 136. Micron* ZBT 8-Word Read



The above timing diagram shows an 8-word read burst to a 32-bit Micron ZBT device. Since the Expansion bus controller supports 8-word wrap transfers, the address shown above may wrap around an 8-word boundary. If the burst was a 16-bit Micron ZBT device, the burst would have twice the number of data cycles.



12.4.1.8.9 Micron* ZBT 4-Word Write Access

Figure 137. Micron* ZBT 4-Word Write

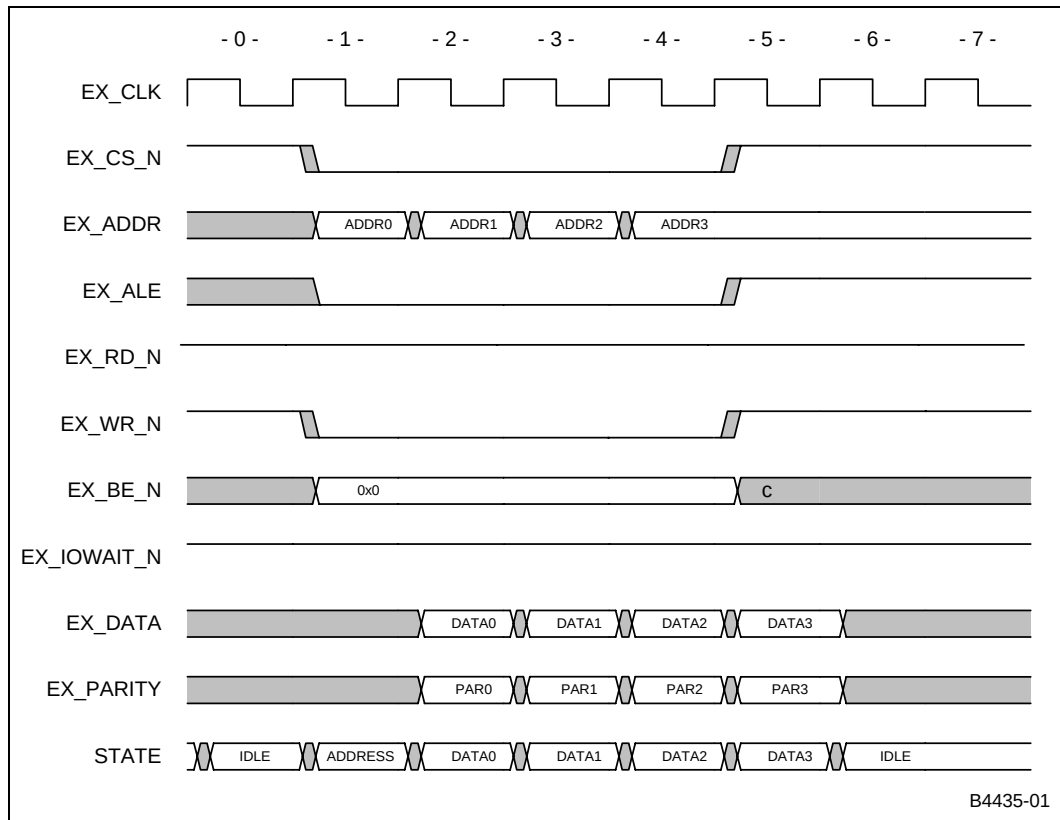
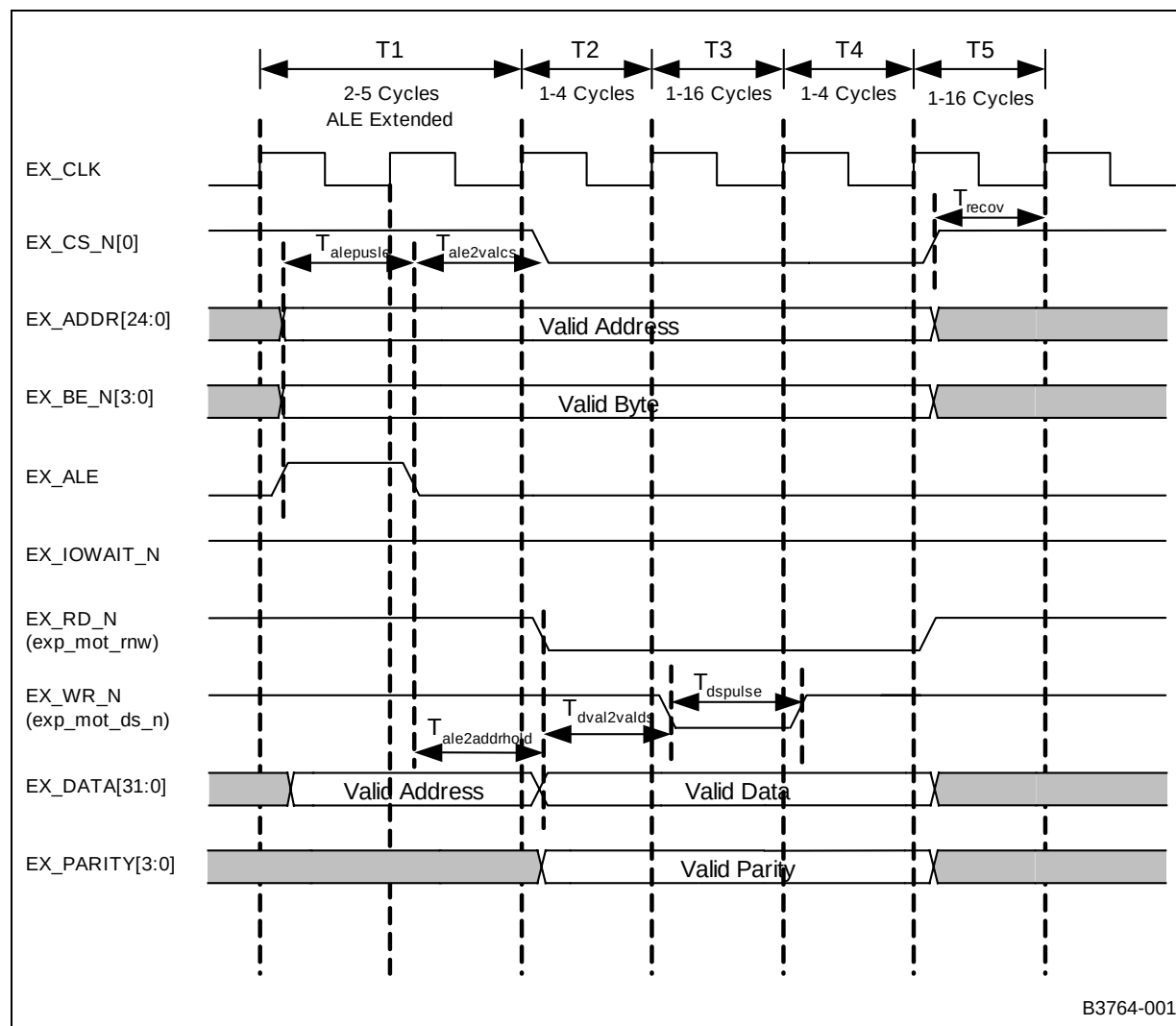


Figure 137 shows a 4-word write burst to a 32-bit Micron ZBT device. Burst writes are only supported with 32-bit Micron ZBT devices.



12.4.1.8.10 Motorola*, Multiplexed-Mode Write Access

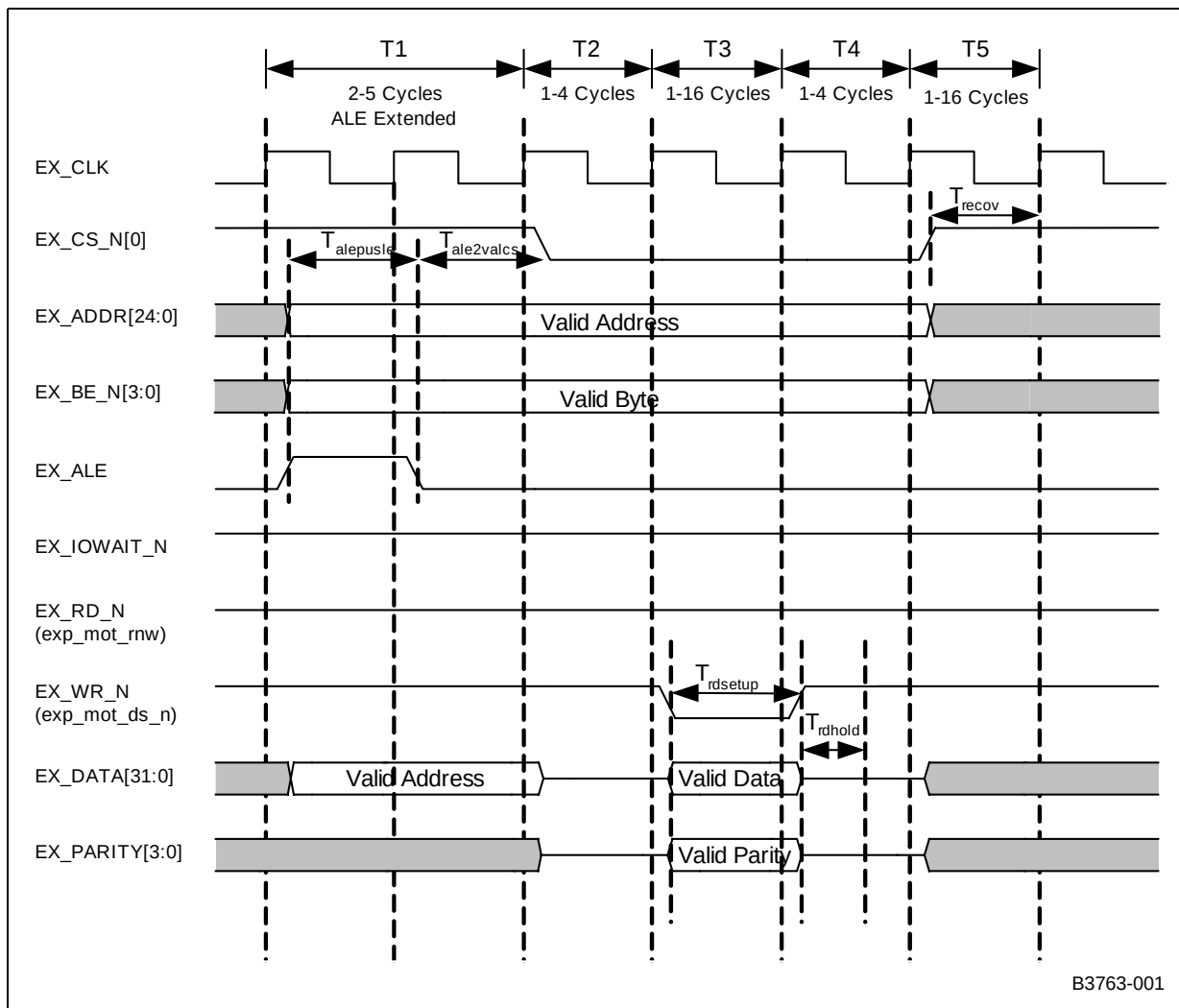
Figure 138. Expansion Bus Write (Motorola*, Multiplexed Mode)





12.4.1.8.11 Motorola*, Multiplexed-Mode Read Access

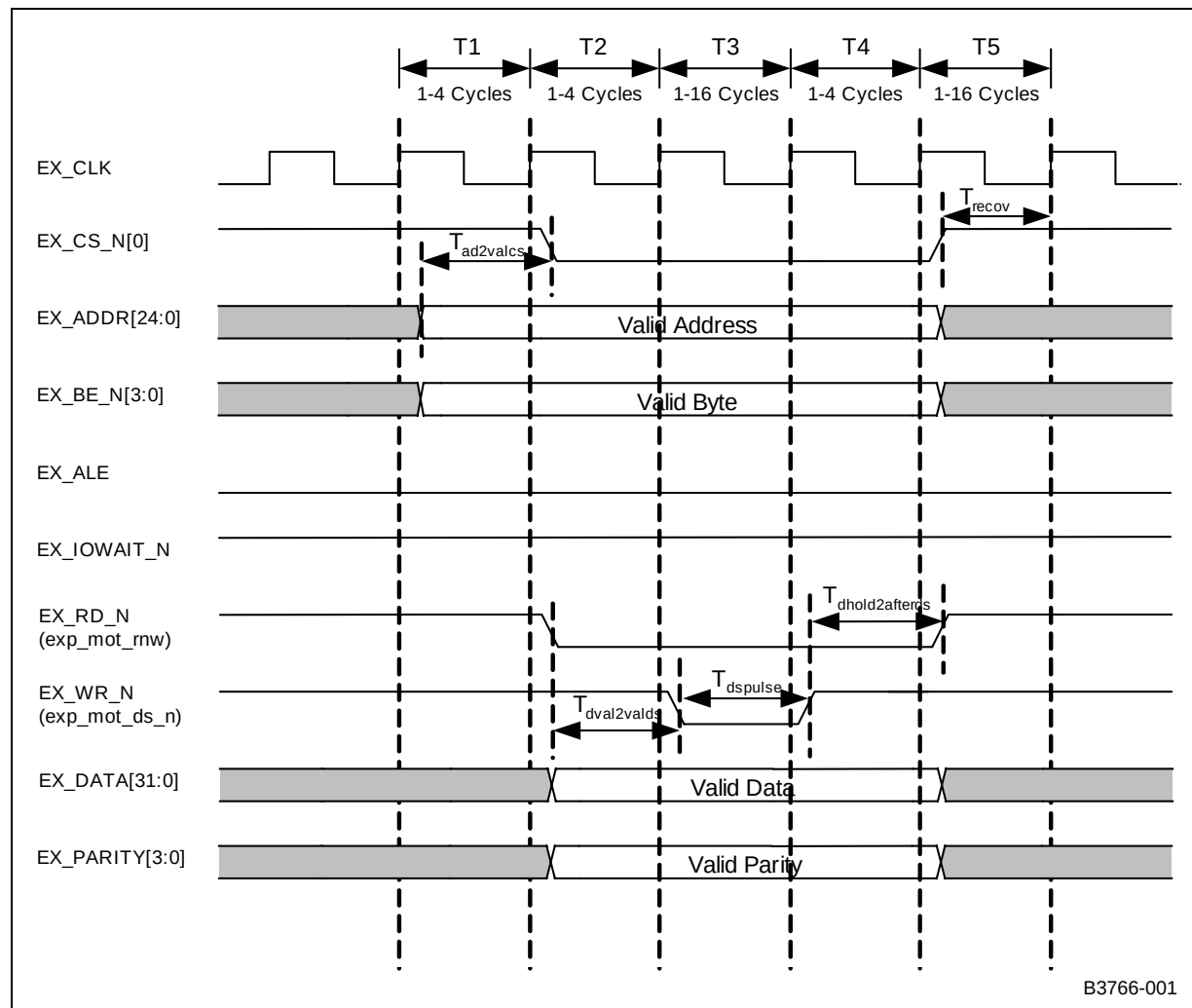
Figure 139. Expansion Bus Read (Motorola*, Multiplexed Mode)





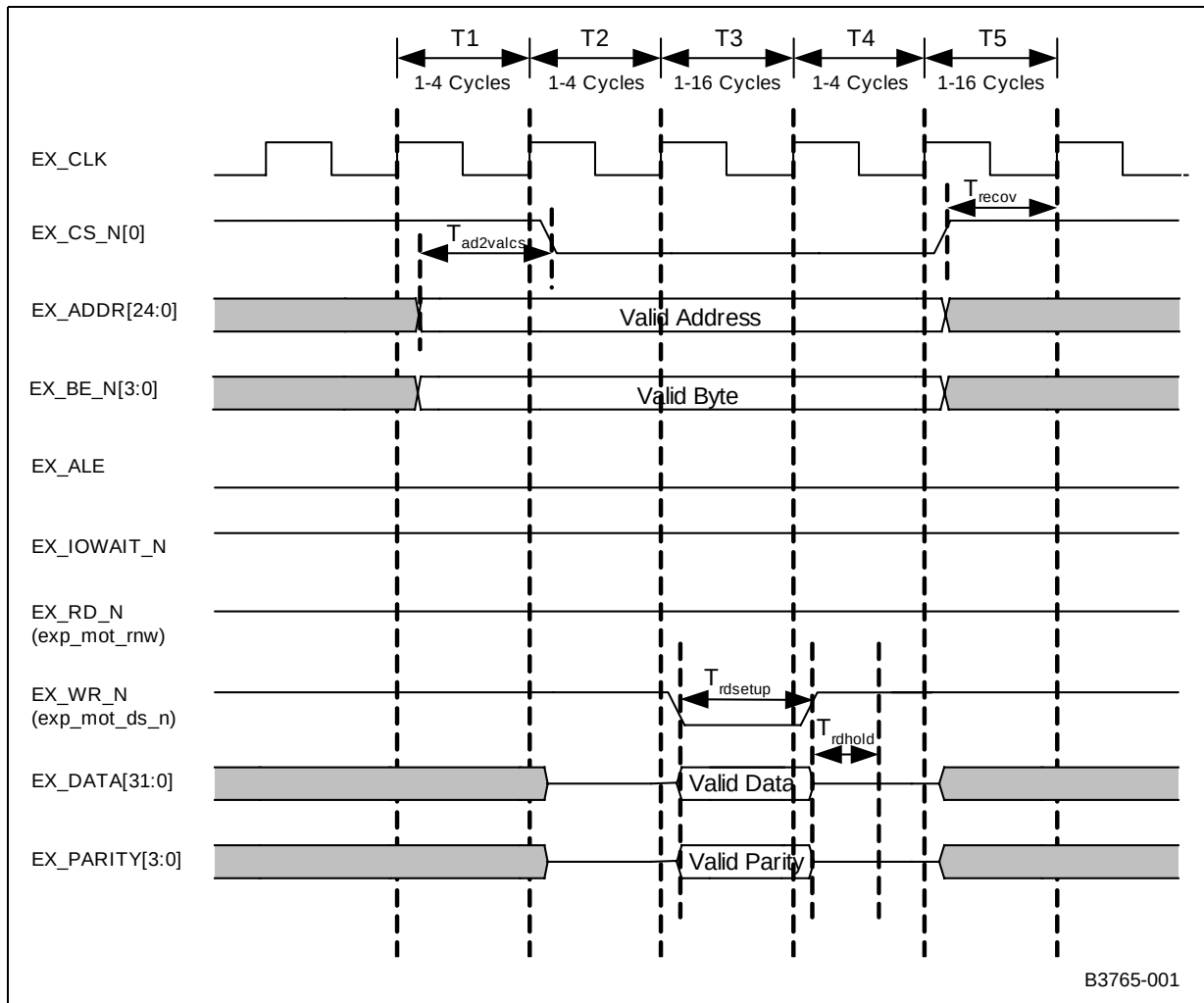
12.4.1.8.12 Motorola*, Simplex-Mode Write Access

Figure 140. Expansion Bus Write (Motorola*, Simplex Mode)



12.4.1.8.13 Motorola*, Simplex-Mode Read Access

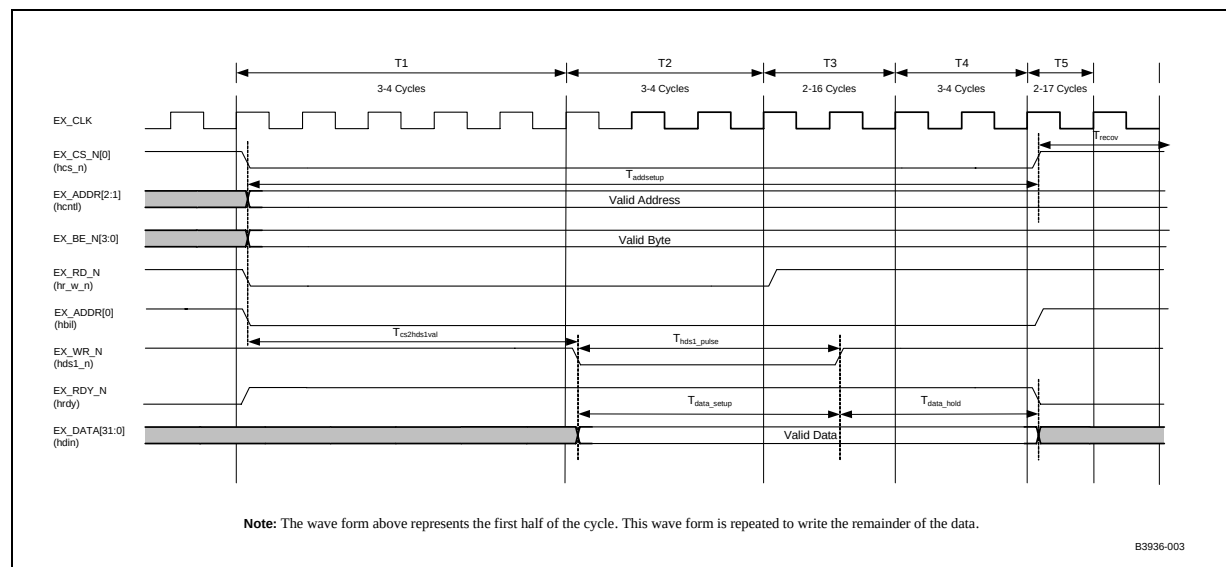
Figure 141. Expansion Bus Read (Motorola*, Simplex Mode)





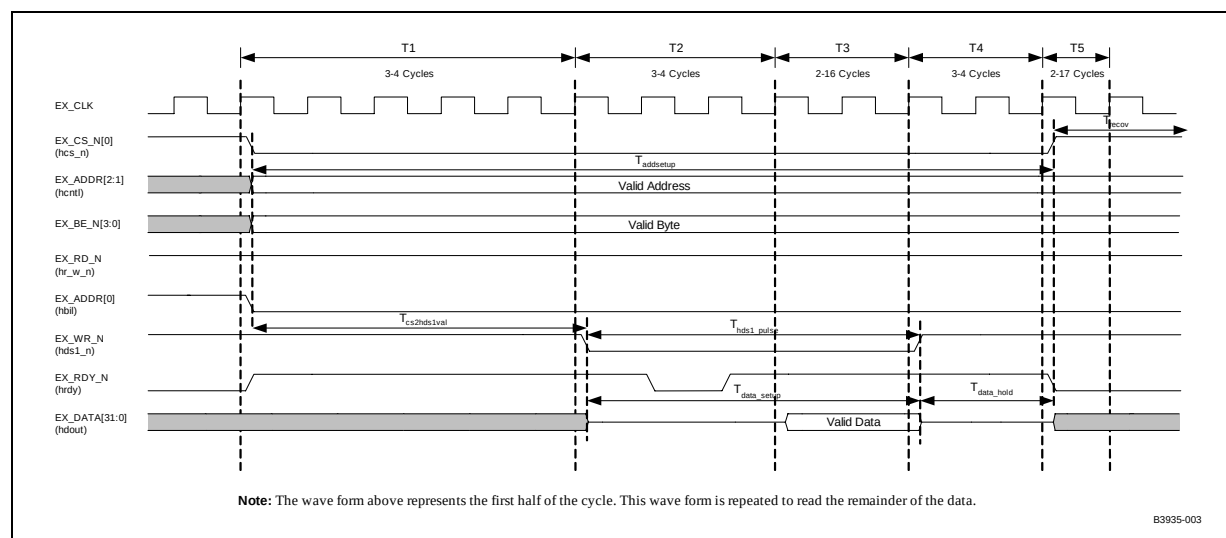
12.4.1.8.14 TI* HPI-8 Write Access

Figure 142. Expansion Bus Write (TI* HPI-8 Mode)



12.4.1.8.15 TI* HPI-8 Read Access

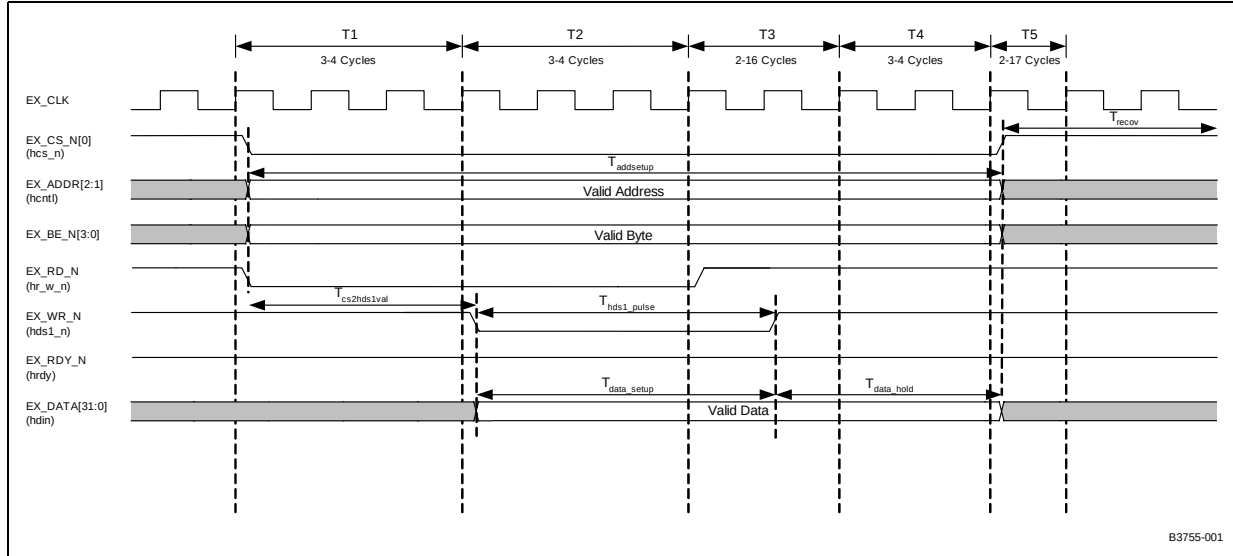
Figure 143. Expansion Bus Read (TI* HPI-8 Mode)





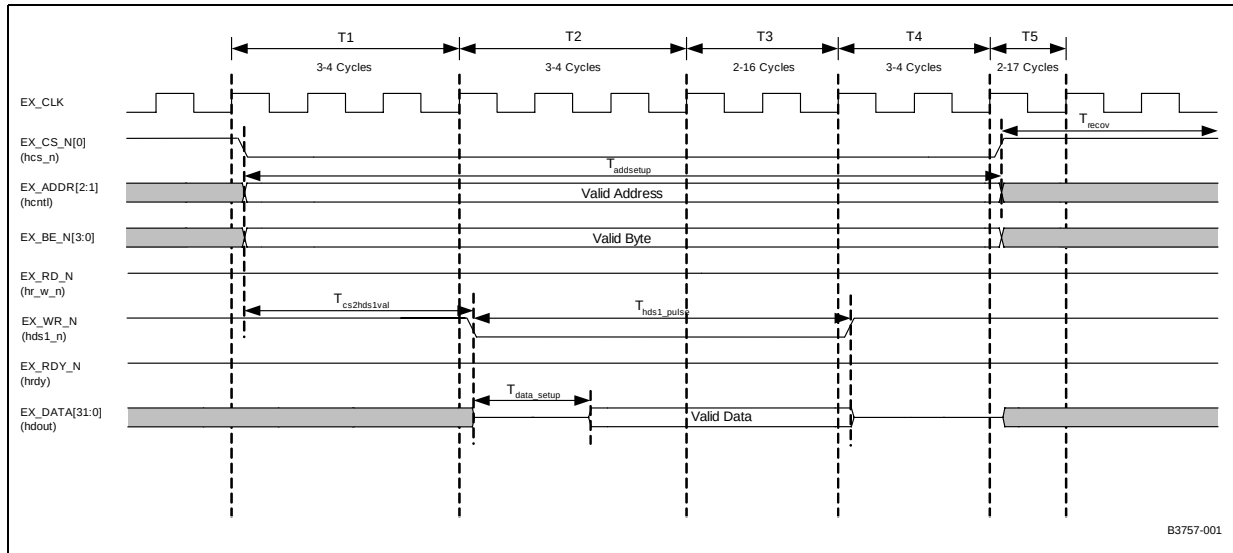
12.4.1.8.16 TI* HPI-16, Multiplexed-Mode Write Access

Figure 144. Expansion Bus Write (TI* HPI-16, Multiplexed Mode)



12.4.1.8.17 TI* HPI-16, Multiplexed-Mode Read Access

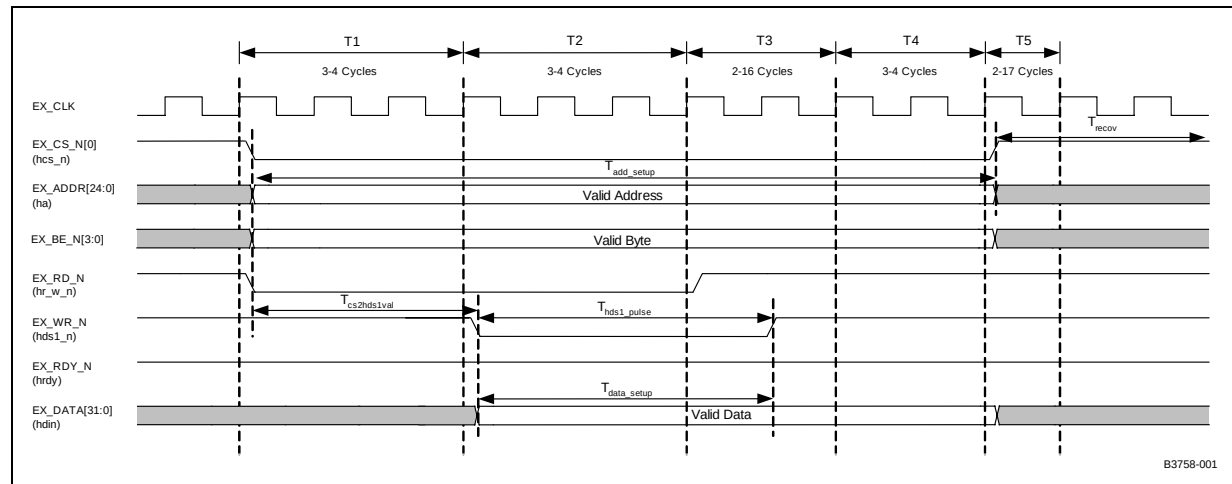
Figure 145. Expansion Bus Read (TI* HPI-16, Multiplexed Mode)





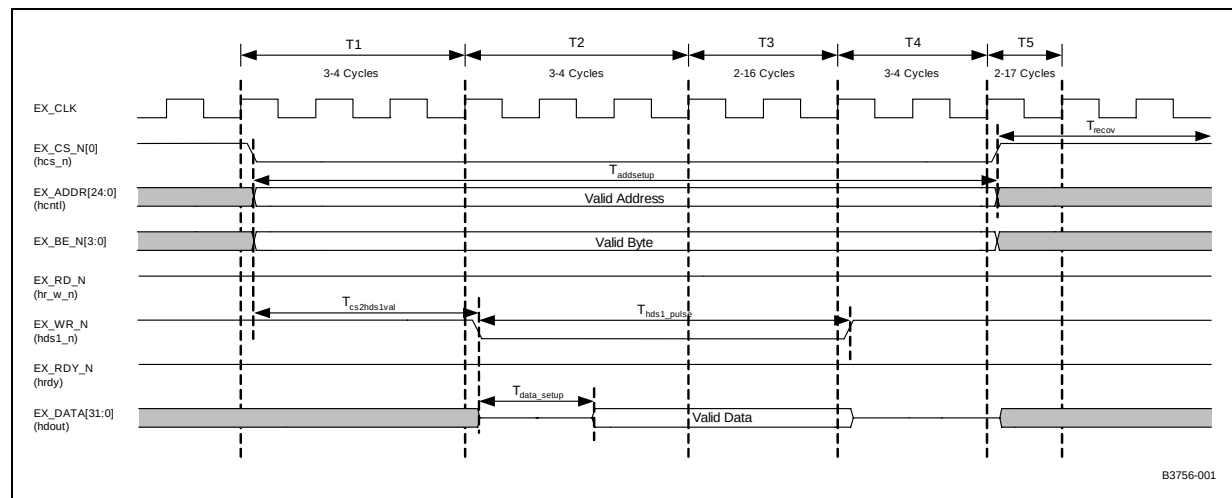
12.4.1.8.18 TI* HPI-16, Simplex-Mode Write Access

Figure 146. Expansion Bus Write (TI* HPI-16, Simplex Mode)



12.4.1.8.19 TI* HPI-16, Simplex-Mode Read Access

Figure 147. Expansion Bus Read (TI* HPI-16, Simplex Mode)



12.4.2 Inbound Transfers

Inbound transfers are initiated by external masters and target IXP45X/IXP46X network processors when EX_SLAVE_CS_N is asserted. The Expansion bus controller should ignore all control signals when EX_SLAVE_CS_N is not asserted and never drive EX_WAIT_N/EX_DATA/EX_PARITY when its not being accessed. External masters must request the Expansion bus and only own the bus when it is granted. Refer to [“Arbitration” on page 686](#) for more information.

The Expansion bus controller has a 1-deep inbound queue that contains one address register and 8 data registers. The Expansion bus only supports 32-bit masters which means no byte lane shifting hardware is needed. Therefore bits 31:0 of EX_DATA bus always translates to bits 31:0 of the AHB DATA bus when EXP_BYTE_SWAP_EN is disabled. The Expansion bus controller supports word aligned read transfers for 1-word



and 8-word aligned read transfers for 8-word burst lengths. The Expansion bus controller also supports write data transfers for byte, halfword (half-word aligned), 1-word (word aligned), and 8-word (8-word aligned) burst lengths. The Expansion bus controller determines the burst size based upon EX_BURST and EX_BE_N and a list of supported transfer types are shown in Table 223. For writes EX_ADDR[1:0] is always ignored, since EX_BE_N indicates which bytes need to be written. For 8-word writes the Expansion bus controller ignores EX_BE_N. For reads EX_ADDR[1:0] and EX_BE_N are ignored since the Expansion bus controller does not perform sub-word reads. The Expansion bus controller will have unpredictable results if the external master generates an unsupported data transfer. For 8-word reads and writes, the external master must drive EX_ADDR[4:2] to 0x0 on the first address cycle of the burst. Also EX_ADDR[24:5] must not change during an 8-word burst. For 8-word writes, the burst ends when EX_ADDR[4:2] = 0x7. For writes, EX_ADDR[4:2] is increment by 0x1 after each word is transferred. Reads do not require all 8 words to be transferred. EX_DATA always lags EX_ADDR by one cycle for reads and writes.

Table 223. Supported Inbound Expansion Bus Transfers

Expansion Bus Cycle	EX_BURST	EX_BE_N	AHB Address Value (AHB_ADDR[1:0])	AHB_HSIZE	AHB_HBURST	Data Translation between EX_DATA and AHB_DATA (EXP_BYTE_SWAP_EN = 0)
Byte write	0	0xE	11	byte	single	EX_DATA[7:0] = AHB_DATA[7:0]
Byte write	0	0xD	10	byte	single	EX_DATA[15:8] = AHB_DATA[15:8]
Byte write	0	0xB	01	byte	single	EX_DATA[23:16] = AHB_DATA[23:16]
Byte write	0	0x7	00	byte	single	EX_DATA[31:24] = AHB_DATA[31:24]
Halfword write	0	0xC	10	halfword	single	EX_DATA[15:0] = AHB_DATA[15:0]
Halfword write	0	0x3	00	halfword	single	EX_DATA[31:16] = AHB_DATA[31:16]
1-word write	0	0x0	00	word	single	EX_DATA[31:0] = AHB_DATA[31:0]
8-word write	1	X	00	word	incr8	EX_DATA[31:0] = AHB_DATA[31:0]
1-word read	0	X	00	word	single	EX_DATA[31:0] = AHB_DATA[31:0]
8-word read	1	X	00	word	incr8	EX_DATA[31:0] = AHB_DATA[31:0]

Read transactions start with EX_SLAVE_CS_N and EX_RD_N asserted. The cycle after EX_SLAVE_CS_N is asserted, the Expansion bus controller will always assert EX_WAIT_N since it has to obtain the read data from the AHB bus. For read data transfers, the Expansion bus controller starts to drive EX_DATA (with invalid data) two cycles after the assertion of EX_SLAVE_CS_N. After the Expansion bus controller receives the data from the AHB bus, it deasserts EX_WAIT_N and drives EX_DATA with valid data. A state machine showing the Expansion bus inbound states is shown in Figure 148. The Expansion bus controller will never assert EX_WAIT_N again after deasserting EX_WAIT_N, unless a new transfer is started.

As soon as the master samples EX_WAIT_N deasserted it can end the transfer for 1-word read by deasserting EX_SLAVE_CS_N. The master can then leave EX_SLAVE_CS_N deasserted for any length of cycles and start a new transaction by asserting EX_SLAVE_CS_N. Alternatively, as soon as the master samples EX_WAIT_N deasserted the master can start a new read transfer by never deasserting



EX_SLAVE_CS_N but deasserting EX_RD_N for at least one cycle. The new transfer starts when EX_RD_N is asserted and the Expansion bus controller will always assert EX_WAIT_N one cycle later.

For 8-word read transfers, the Expansion bus protocol begins the same way as a 1-word transfer, however data words 2 through 8 will never assert EX_WAIT_N. Typically the master increments EX_ADDR[4:2] by 0x1 for each data cycle until all 8 words are transferred, however this is not mandatory. The Expansion bus controller only monitors EX_ADDR[4:2] to determine which word is being transferred. For 8-word data transfers, external masters can also insert wait states for each word of the data transfer by not incrementing EX_ADDR every cycle. Once all 8 words have been transferred the master can deassert EX_SLAVE_CS_N. For 8-word reads, the master can also end the burst early by deasserting EX_SLAVE_CS_N anytime after EX_WAIT_N is deasserted. The master can then re-start the 8-word read by re-asserting EX_SLAVE_CS_N, however the Expansion bus controller will not assert EX_WAIT_N if EX_ADDR[4:2] is not equal to 0x0. This will significantly improve the performance of 8-word reads, since the data is already present in the Expansion bus data fifo.

The master can also choose to do back-to-back 8-word reads by leaving EX_SLAVE_CS_N asserted but deasserting EX_RD_N for at least one cycle. The Expansion bus controller will monitor EX_ADDR[4:2] to determine when a new 8-word burst has started and assert EX_WAIT_N one cycle after the assertion of EX_RD_N. EX_WAIT_N will be deasserted when the Expansion bus controller has received the data from the AHB.

The Expansion bus controller does not support back-to-back reads of different burst lengths; the master must always deassert EX_SLAVE_CS_N between different burst length reads. Additionally the Expansion bus controller does not support back-to-back reads followed by writes or writes followed by reads. Unpredictable operation will result if EX_BURST is changed while EX_SLAVE_CS_N is asserted for reads or writes.

Write transactions start with EX_SLAVE_CS_N and EX_WR_N asserted. The Expansion bus controller ignores EX_ADDR on the first cycle of a new write transfer. For write transactions the Expansion bus controller will need to assert EX_WAIT_N only if its write queue is busy completing a previous write transfer. If the Expansion bus inbound queue is free, the Expansion bus controller will not assert EX_WAIT_N for the current write transfer. Once the master detects that EX_WAIT_N is deasserted, the master can deassert EX_SLAVE_CS_N for 1-word/sub-word data transfers. The master can also start a new write transfer by leaving EX_SLAVE_CS_N asserted but deasserting EX_WR_N for at least one cycle. The new transfer starts when EX_WR_N is asserted and the Expansion bus controller will assert EX_WAIT_N one cycle later if its busy completing the previous write.

For 8-word write transfers, the Expansion bus protocol begins the same way as a 1-word transfer. The external master cannot start the 2nd address cycle until EX_WAIT_N is sampled deasserted. Since it takes one cycle for the Expansion bus controller to drive EX_WAIT_N, the first address cycle will always be at least two cycles. However, data words 2 through 8 will never assert EX_WAIT_N. For 8-word writes, the master increments EX_ADDR[4:2] by 0x1 for each data cycle until all 8 words are transferred. The Expansion bus controller monitors EX_ADDR[4:2] to determine which word is being transferred. The Expansion bus controller samples EX_DATA when EX_ADDR[4:2] increments, a NOP cycle is inserted after a data state, or when EX_SLAVE_CS_N is deasserted. For 8-word data transfers, external masters can also insert wait states for each word of the data transfer by not incrementing EX_ADDR every cycle, however the setup and hold times must be met for EX_ADDR on every cycle. For 8-word writes, the master can temporarily end the burst anytime after EX_WAIT_N is deasserted by deasserting EX_SLAVE_CS_N. The master must re-start the 8-word write by re-asserting EX_SLAVE_CS_N to complete the burst, however the Expansion bus controller will not assert EX_WAIT_N if EX_ADDR[4:2] is not equal to 0x0. The Expansion bus controller will only send the 8-word write data to the AHB when EX_ADDR[4:2] = 0x7 and the master has finished the transfer by deasserting



EX_SLAVE_CS_N or EX_WR_N. The master must transfer all 8-words during an 8-word write burst. The master can also choose to do back-to-back 8-word writes by leaving EX_SLAVE_CS_N asserted but deasserting EX_WR_N for at least one cycle. The Expansion bus controller will monitor EX_ADDR[4:2] to determine when a new 8-word burst has started and assert EX_WAIT_N one cycle after the assertion of EX_WR_N. EX_WAIT_N will deassert when the Expansion bus controller completes the previous write.

EX_WAIT_N will always be deasserted and not driven the cycle after EX_SLAVE_CS_N is deasserted. EX_WAIT_N will have about a 50-K Ω pull-up enabled in the PAD at all times to prevent floating of this signal.

External masters can also choose to insert NOP cycles (EX_RD_N = '1', EX_WR_N = '1') during a data transfer. The Expansion bus controller will always ignore EX_ADDR, EX_BE_N, and EX_BURST during NOP cycles. Adding NOPs in the middle of a burst doesn't shorten the number of valid data phases; even with NOPs the master still needs to perform 8 valid data transfers during write transfers. If a master starts a transfer with NOP, the Expansion bus controller will never assert EX_WAIT_N while EX_RD_N and EX_WR_N are both deasserted. If the master starts the transaction with NOP cycles, the Expansion bus controller will drive EX_WAIT_N the cycle after the first valid write or read cycle. This first write or read cycle will always be at least 2 cycles long, since the external master needs to determine if EX_WAIT_N is asserted. NOPs can be inserted anytime during the data transfer, including between data transfers in a 8-word transfer. However, the external master cannot start a NOP cycle if EX_WAIT_N is asserted or after the first cycle of a read or write cycle, since the Expansion bus controller may assert EX_WAIT_N in this cycle. Additionally the external master must meet setup and hold on EX_ADDR (this means the external master cannot drive EX_ADDR to X) for reads and writes on the data cycle following a NOP cycle during 8-word transfers.

Accesses with EX_RD_N and EX_WR_N both asserted result in unpredictable operation.



Figure 148. Expansion Bus Inbound State Diagram

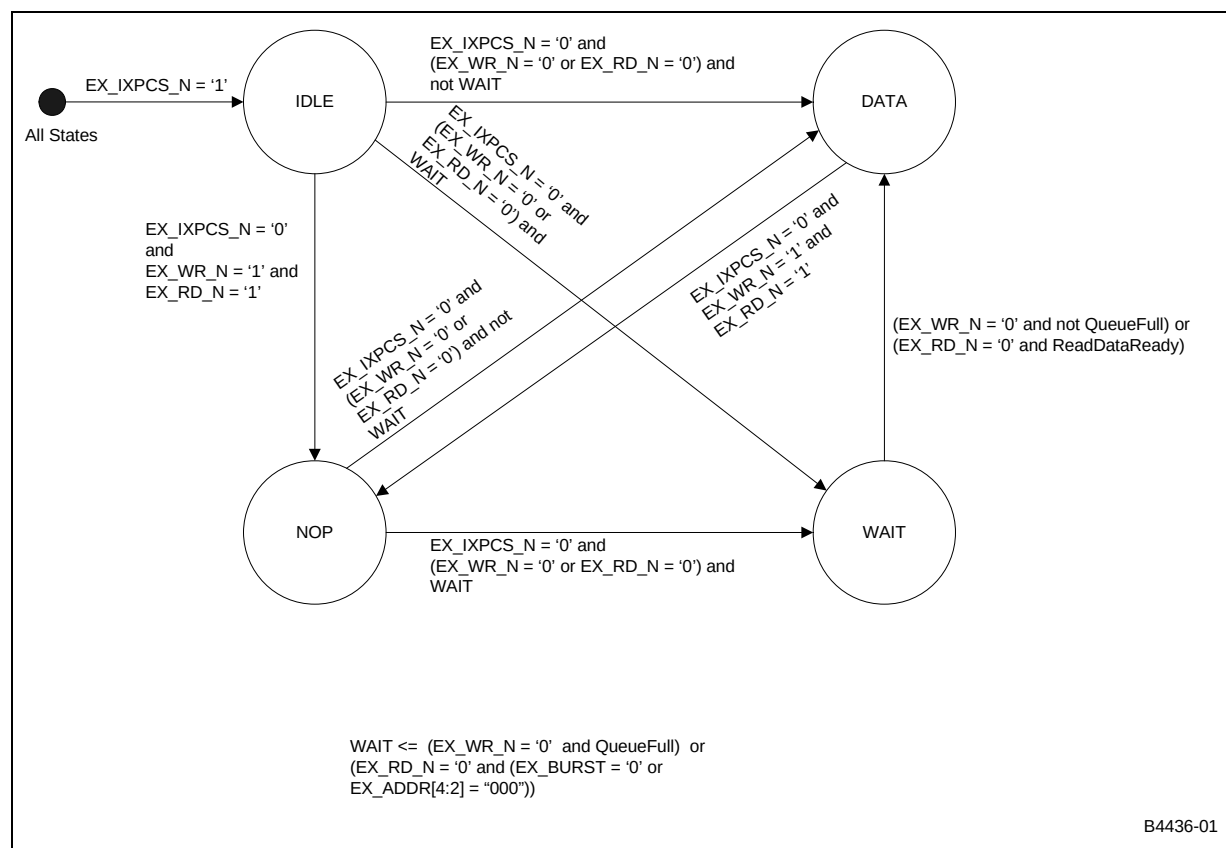


Table 224. EX_ADDR Operation

Command	EX_BURST	EX_ADDR[4:2]	Operation
1-WORD WRITE	0	X	Assert EX_WAIT_N if QueueFull = '1'. Set QueueFull. Transfer Word to AHB. Clear QueueFull when word is transferred.
8-WORD WRITE	1	0x0	Assert EX_WAIT_N if QueueFull = '1'. Write word 0 to data fifo.
8-WORD WRITE	1	0x1	Write word 1 to data fifo.
8-WORD WRITE	1	0x2	Write word 2 to data fifo.
8-WORD WRITE	1	0x3	Write word 3 to data fifo.
8-WORD WRITE	1	0x4	Write word 4 to data fifo.
8-WORD WRITE	1	0x5	Write word 5 to data fifo.
8-WORD WRITE	1	0x6	Write word 6 to data fifo.
8-WORD WRITE	1	0x7	Write word 7 to data fifo. Set QueueFull. Transfer all 8-words to AHB. Clear QueueFull when all 8-words are transferred.
1-WORD READ	0	X	Assert EX_WAIT_N. Request 1-word from AHB. Deassert EX_WAIT_N when data is retrieved from AHB and driven on EX_DATA.
8-WORD READ	1	0x0	Assert EX_WAIT_N. Request 8-words from AHB. Deassert EX_WAIT_N when all 8-words are retrieved from AHB. Drive EX_DATA with word 0.
8-WORD READ	1	NOT (0x0)	Drive EX_DATA with word EX_ADDR[4:2]



Since the AC timings for EX_WAIT_N is critical, the Expansion bus controller will only use EX_SLAVE_CS_N, EX_WR_N, EX_RD_N, EX_ADDR[4:2], EX_BURST, and the Expansion bus controller internal state to determine if it needs to assert EX_WAIT_N. The tri-state enable for EX_DATA pins should be based upon EX_SLAVE_CS_N and EX_RD_N.

For inbound transfers, the AHB address is determined from EX_ADDR and the EXP_INBOUND_ADDR register and is described in detail in [“EXP_INBOUND_ADDR” on page 717](#). For reads the master must drive EX_ADDR on the cycle that EX_SLAVE_CS_N is asserted. For 1-word or sub-word write data transfers, EX_ADDR is only sampled one cycle before EX_SLAVE_CS_N is deasserted. For 8-word writes, the master must drive EX_ADDR the cycle after EX_SLAVE_CS_N is asserted. The AHB bus is configured to be big-endian. The Expansion bus controller uses EX_BE_N to generate sub-word addresses on writes. Sub-word reads are unsupported and the AHB address[1:0] is always 0x0 for reads.

Inbound transfers to the AHB bus can be byte swapped based upon the value of EXP_BYTE_SWAP_EN. If enabled, BYTE 0 is swapped with BYTE 3 and BYTE 1 is swapped with BYTE 2. The byte swapping occurs for all accesses except accesses to Expansion bus MMRs.

12.4.2.1 Parity

The Expansion bus controller generates even or odd parity for each byte returned to EX_DATA and compares parity for each byte written on EX_DATA, if INPAR_EN is set in the EXP_MST_CONTROL register. EX_PARITY is transferred in the same clock cycle in which EX_DATA is transferred. If a write compare results in a parity mismatch on EX_PARITY, the AHB address is logged in the EXP_PARITY_STATUS and InErrorSts is set. Even if there is a parity error, the write data is still transferred to the AHB interface. Exp_parity_error will be asserted by the Expansion bus controller during a parity error and an interrupt will be generated if enabled in the interrupt controller. Exp_parity_error will remain asserted until software clears the EXP_PARITY_STATUS register.

Even parity is defined as the number of 1's on EX_DATA[7:0] and EX_PARITY[0] must be an even number. Parity for the 2nd byte of EX_DATA is generated on EX_PARITY[1]; the 3rd byte of EX_DATA is generated on EX_PARITY[2], and the 4th byte of EX_DATA is generated on EX_PARITY[3]. For byte and halfword writes the Expansion bus controller will not generate or compare parity on byte enables that are not asserted. If INPAR_EN is cleared, the Expansion bus controller doesn't generate or compare parity and EX_PARITY should not toggle to conserve power. Odd parity can be enabled by setting OddPar in the EXP_MST_CONTROL register and is equivalent to the inverted value of even parity.

If the Expansion bus controller receives an error response on AHB HRESP during a read transfer, the Expansion bus controller will drive 0x0 on EX_DATA and generate incorrect parity by inverting EX_PARITY for the offending word/words. If the Expansion bus controller is reading from DDR1 SDRAM and un-correctable ECC error occurs, the Expansion bus controller will also generate incorrect parity on EX_PARITY.

12.4.3 Arbitration

The Expansion bus controller provides arbitration between IXP45X/IXP46X network processors and up to four external masters. An external arbiter can also be used, if desired. The external arbiter is enable by the EX_ADDR driving bit 6 of EX_ADDR to logic 0.



12.4.3.1 Internal Arbitration

The request signal is called EX_REQ_N/EX_REQ_GNT_N and the grant signal is called EX_GNT_N/EX_GNT_REQ_N. The arbitration is a round-robin mechanism that also supports priority of an external master grant and masking requests of external masters. The arbiter will perform a round-robin arbitration cycle when the Expansion bus is idle and not in an active T1-address timing or T5-recovery phase if the Expansion bus controller has bus ownership. During the arbitration cycle, the arbiter will grant the highest priority master. If an external master is granted the bus and GrantRemove in the EXP_MST_CONTROL register is clear, the external master will not lose grant until that master deasserts request. If the external master keeps its request asserted, it can transfer several separate data transfers without ever losing the Expansion Bus. After that master deasserts request, the arbiter performs a round-robin arbitration cycle and grant the next requesting master. The GrantRemove bit must be static once configured.

External masters can assert and deassert request in any cycle. However, the arbiter will only assert grant to a new requesting master when the Expansion bus is idle (EX_SLAVE_CS_N and EX_CS_N deasserted). This will guarantee that a data transfer will not be interrupted before its complete. The arbiter will always assert grant for at least two cycles. This allows a master one extra cycle to start a transaction without risking losing the ownership of the Expansion bus. However a master can only start a data transfer if its grant is asserted in the previous cycle. Refer to the arbitration timing diagrams in [“Expansion Bus Arbiter Timing Diagrams” on page 698](#) for more details.

If GrantRemove is set, an external master may lose grant in any cycle, however that master still owns the bus until EX_SLAVE_CS_N and EX_CS_N are deasserted. The external master must start a data transfer within two cycles of acknowledging grant or risk losing bus ownership. If the master loses grant, the master cannot start a new transaction.

The arbiter also supports masking of all 4 external requests if ArbMask# in the EXP_MST_CONTROL register is clear. This bit can be set and cleared at any time. If an external master is granted the bus while ArbMask# is being cleared, that external master will not lose grant until it deasserts its request if GrantRemove is clear or until the Expansion bus is idle if GrantRemove is set. If ArbMask# is clear and once the IXP45X/IXP46X network processors are granted the Expansion bus, the processors will not lose the Expansion bus until ArbMask# is set by software. This functionality allows atomic accesses by the Intel XScale processor and allows higher performance for critical outbound accesses to the Expansion bus.

After reset, the arbiter will not start arbitration until the ArbMask# is set by software.

When no bus masters are requesting the Expansion bus, the arbiter will park grant on IXP45X/IXP46X network processors. Park grant means that grant will be awarded to the IXP45X/IXP46X network processors. When an external master is granted the bus, the IXP45X/IXP46X network processors will tri-state the following shared Expansion bus signals: EX_ALE, EX_ADDR, EX_BE_N, EX_CS_N, EX_DATA, EX_PARITY, EX_RD_N, EX_WR_N.

External board pull-ups are required on EX_CS_N to ensure this signal remains deasserted so that a spurious transfer does not start due to board noise. Additionally, the system designer is responsible for ensuring that all the tri-stated signals do not become indeterminate. If they become indeterminate, excessive power consumption will occur in the PAD input buffers. When the IXP45X/IXP46X network processors are granted or parked on the Expansion bus, the processors will drive the shared Expansion bus signals to a de-asserted or stable state to minimize power consumption. Due to the arbitration between the IXP45X/IXP46X network processors and external masters, the Expansion bus will float for a few cycles during the arbitration; however this typically is not a problem due to the capacitance on the board and the required pull-ups on EX_CS_N.



12.4.3.2 External Arbiter

If the external arbiter is enabled, the Expansion bus controller request line is called EX_GNT_REQ_N and its grant signal is called EX_REQ_GNT_N. EX_REQ_N is ignored and pull-ups in the PAD I/O block are enabled to avoid a floating input. When the Expansion bus controller needs access to the Expansion bus, the Expansion bus controller will assert request. As soon as the Expansion bus controller is granted the bus, the Expansion bus controller may start a data transfer. The Expansion bus controller will keep the request line asserted until all phases (T1,T2,T3,T3,T5) of the data transfer are complete. If there is not another data transfer to be performed, the Expansion bus controller will deassert request. After the Expansion bus controller deasserts request, the external arbiter can deassert grant. The external arbiter must not deassert grant until request is deasserted.

If the external arbiter parks grant on the Expansion bus controller, the Expansion bus controller will not start a data transfer until two cycles after it has asserted request assuming it still is granted the bus. The Expansion bus controller must wait 2 cycles with request asserted to ensure the external arbiter does not remove the grant the same cycle as request is asserted. Refer to [“External Expansion Bus Timing Diagram” on page 700](#) for more details.

12.4.4 Multiple Processors Connected by the Expansion Bus

When connecting multiple IXP45X/IXP46X network processors together via the Expansion bus, a special reset sequence is required to ensure that each of the IXP45X/IXP46X network processors has independent configuration values as defined in [“Configuration Register 0” on page 706](#).

In order to boot multiple IXP45X/IXP46X network processors, the system must assert the RESET_IN_N pin to processors #0 and #1 upon power up. The Expansion bus controller has weak pull-ups on all shared Expansion bus signals to prevent floating during reset. After the board configures the appropriate EX_ADDR configuration straps, RESET_IN_N can be deasserted on processor #0. Processor #0 will begin to boot, however processor #1 will remain in reset. Typically processor #0 will be the Expansion bus arbiter and processor #1 will use processor #0 as the arbiter.

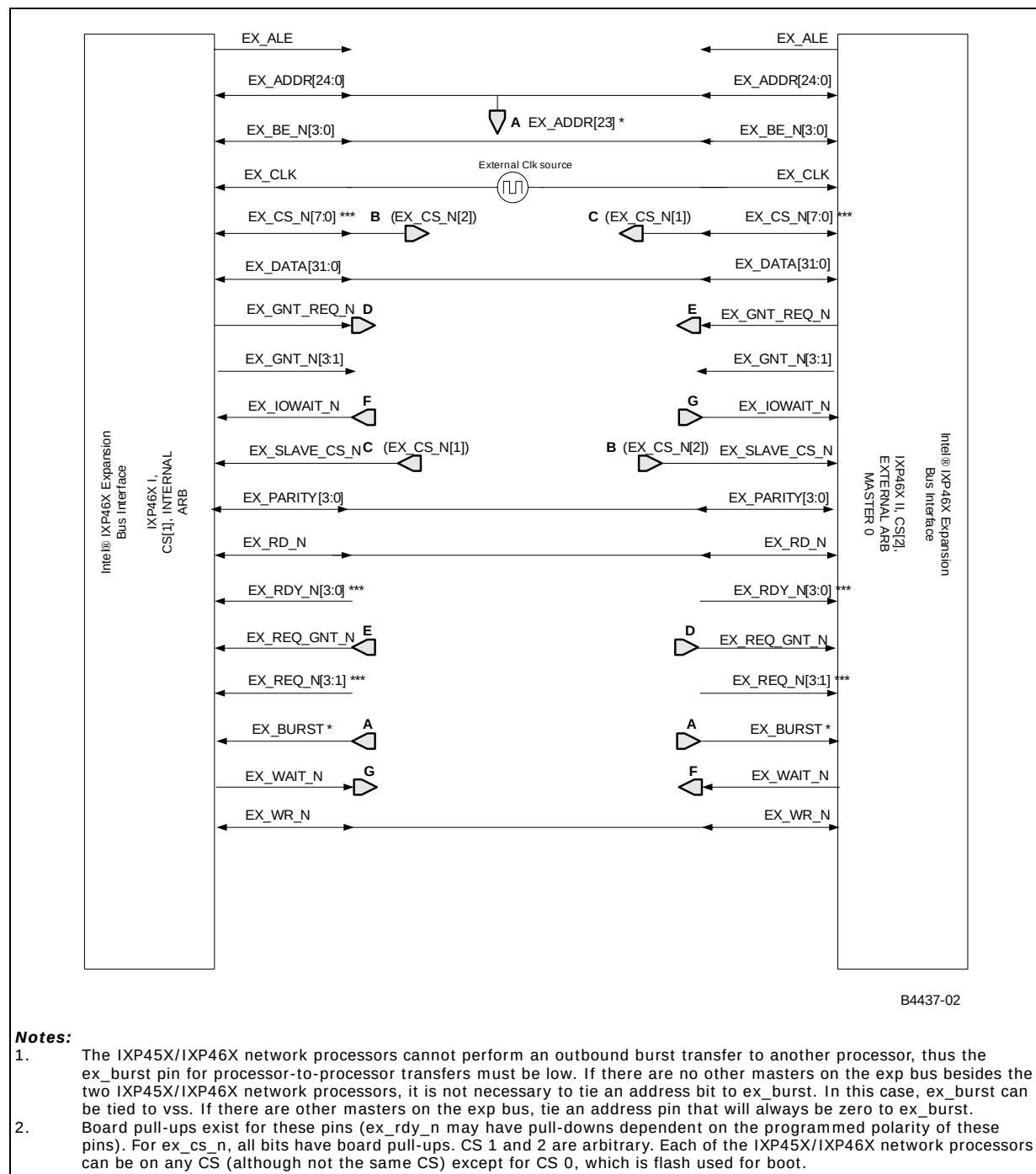
After the boot sequence is complete for processor #0, an outbound read or write needs to be issued to the Expansion bus to force EX_ADDR to the appropriate configuration values for processor #1. Once EX_ADDR is stable, RESET_IN_N can be deasserted for processor #1. The system designer can use a GPIO pin on processor #0 to signal that EX_ADDR is stable.

When programming the EXP_TIMING_CS register to access other IXP45X/IXP46X network processors, T3 must be programmed with a minimum of 3 clocks, T5 must be programmed to either 0, 1, or 2 clocks, WORD_EN must be set to 32-bit bus, MUX_EN must be 0, CYC_TYP must be programmed to Intel Cycles, and EXP_CHIP must be set to 1. T1,T2, and T4 can be programmed to any value, but typically they are left at 0 clocks.

When connecting multiple IXP45X/IXP46X network processors together via the Expansion bus, the GrantRemove bit must be clear. This requirement is necessary to guarantee that the Expansion master will never lose grant during the T1-address timing or T5-recovery phase. [Figure 149](#) shows the connection of multiple IXP45X/IXP46X network processors.



Figure 149. Multiple IXP45X/IXP46X network processors Connected Back-to-Back



B4437-02



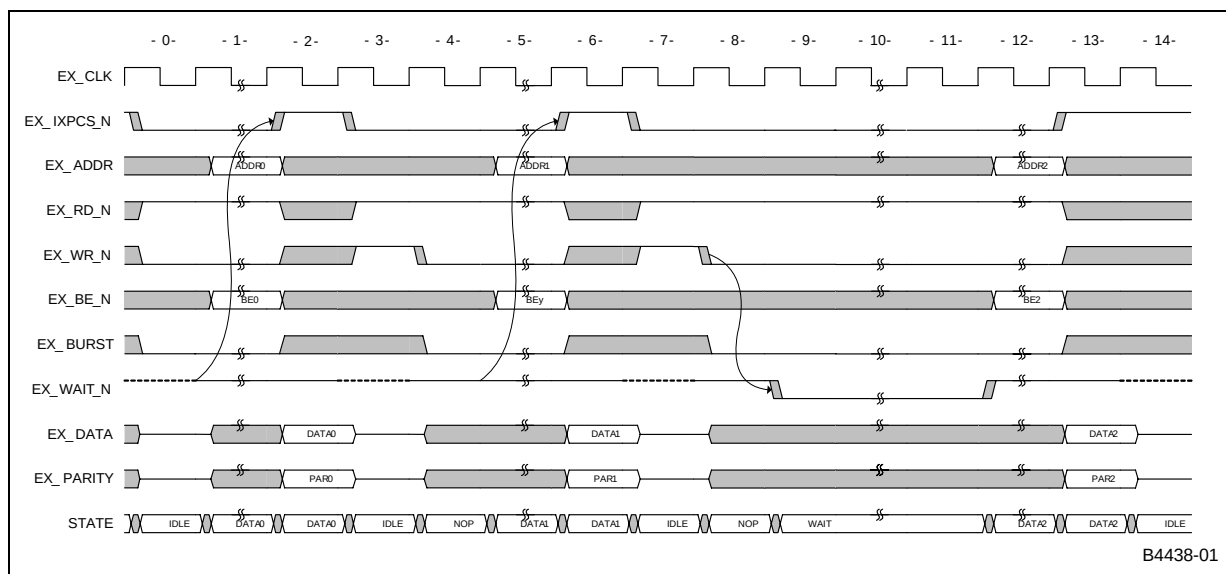
12.4.5 Expansion Bus Inbound Timing Diagrams

The next several timing diagrams show several representations of some of the supported inbound bus protocol. The Expansion bus controller is not limited to the possibilities of the timing diagrams shown. Timing diagrams with varying combinations of NOPS, Master waits, back-to-back transfers are all supported as long as the rules that are described in [Section 12.4.2, “Inbound Transfers” on page 681](#) are followed.

12.4.5.1 Back-to-Back 1-Word Inbound Write with EX_SLAVE_CS_N Deasserted

[Figure 150](#) shows three 1-word writes. The first transaction is a 1-word write with no waits. In the second write the master chose to insert a NOP in cycle 3. In the third write, the Expansion bus controller is busy and asserts EX_WAIT_N in cycle 9. The Expansion bus controller will assert EX_WAIT_N one cycle after the assertion of EX_SLAVE_CS_N and EX_WR_N if its not ready to transfer data. If EX_WAIT_N is asserted, the master cannot end the burst until EX_WAIT_N is deasserted. For 1-word writes, the Expansion bus controller only transfers the data that is presented on the cycle that EX_SLAVE_CS_N is deasserted. In the timing diagram the master chose to tri-state EX_DATA/EX_PARITY in cycles 3 and 7, however this is not mandatory and can be driven.

Figure 150. Back-to-Back 1-Word Inbound Write with EX_SLAVE_CS_N Deasserted

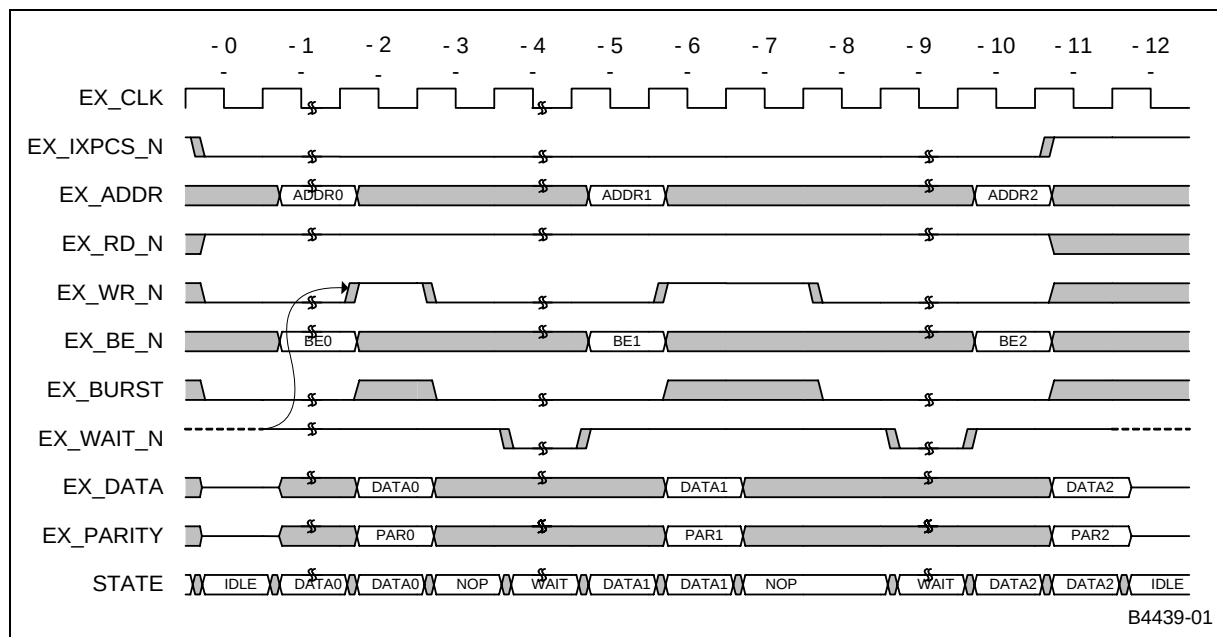


12.4.5.2 Back-to-Back 1-Word Writes without Deasserting EX_SLAVE_CS_N

[Figure 151](#) shows multiple 1-word writes without deasserting EX_SLAVE_CS_N. The external master must deassert EX_WR_N between the writes and the Expansion bus controller will assert EX_WAIT_N once EX_WR_N is re-asserted in cycle 4. When the Expansion bus controller observes EX_WR_N deasserting in cycle 2 and 6, it captures the data.



Figure 151. Back-to-Back 1-Word Writes without Deasserting EX_SLAVE_CS_N

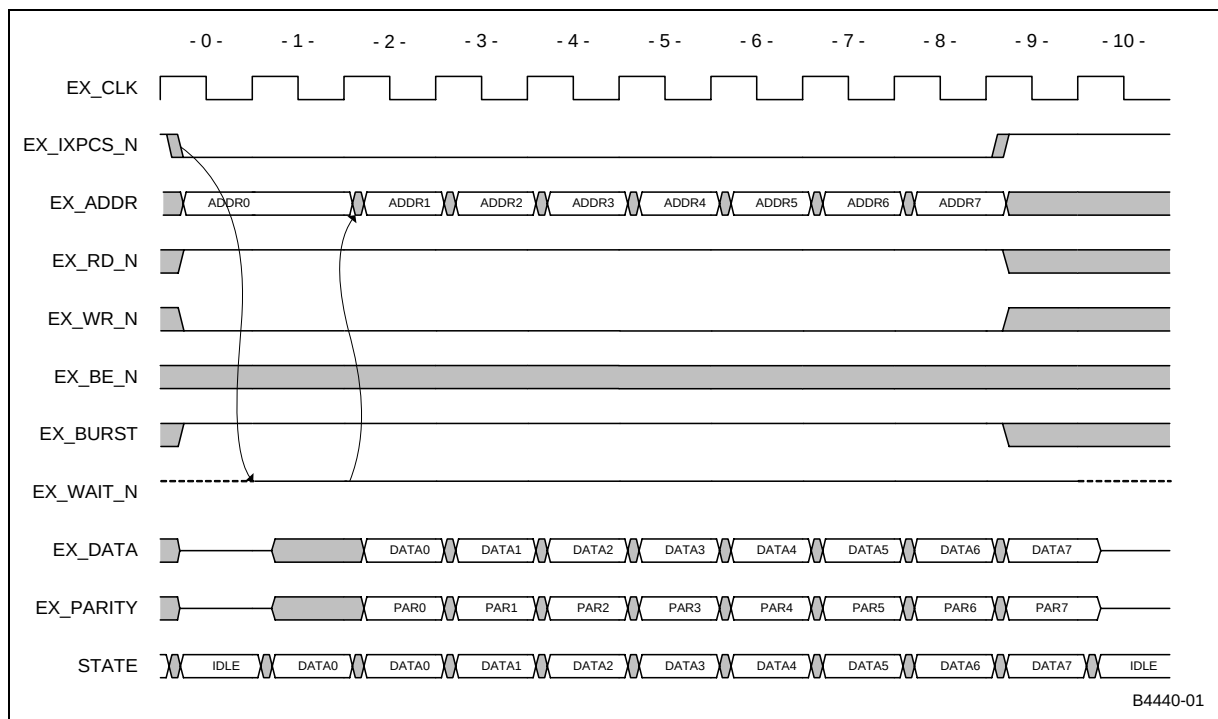


12.4.5.3 Eight-Word Inbound Write

Figure 152 shows an 8-word write with no wait states introduced from the Expansion bus controller. In cycle 2, the external master senses that EX_WAIT_N is not asserted and the address can be incremented.



Figure 152. Eight-Word Inbound Write

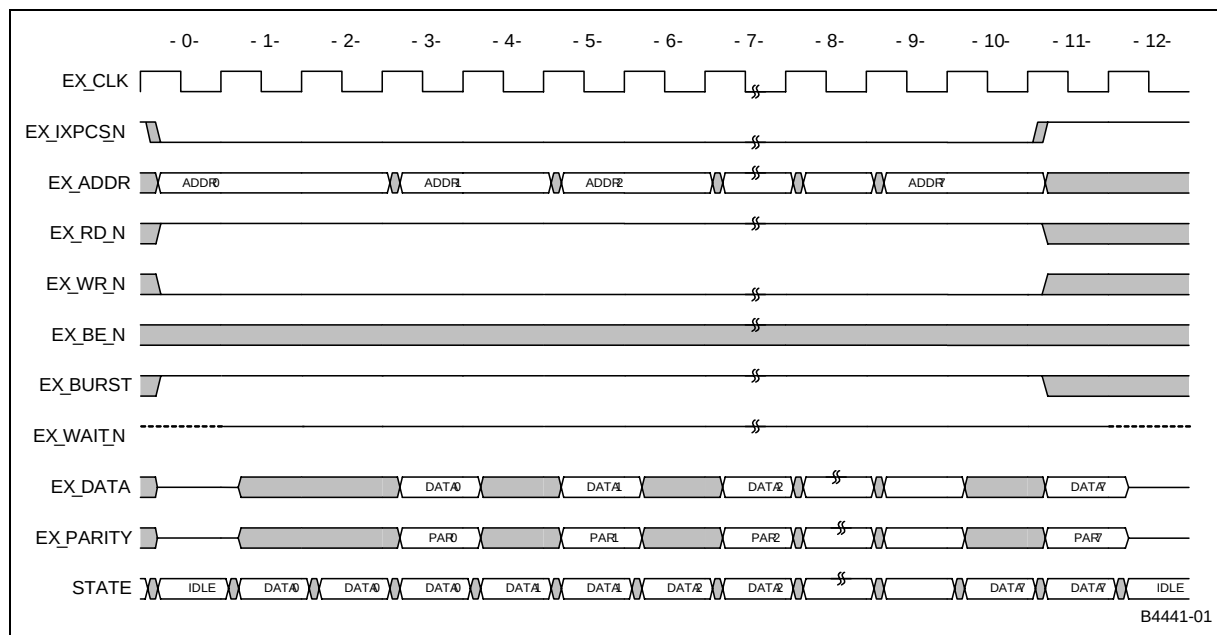


12.4.5.4 Eight-Word Inbound Write with Master Wait States

Figure 153 shows an 8-word write in which the master increments EX_ADDR every other cycle. The Expansion bus controller only captures data on the cycle that EX_ADDR transitions to the next address. The external master can choose to insert as many repeat address cycles as desired, however the setup and hold time of EX_ADDR must be met every cycle.



Figure 153. Eight-Word Inbound Write



12.4.5.5 Eight-Word Inbound Write with NOPS

Figure 154. Eight-Word Inbound Write with NOPS

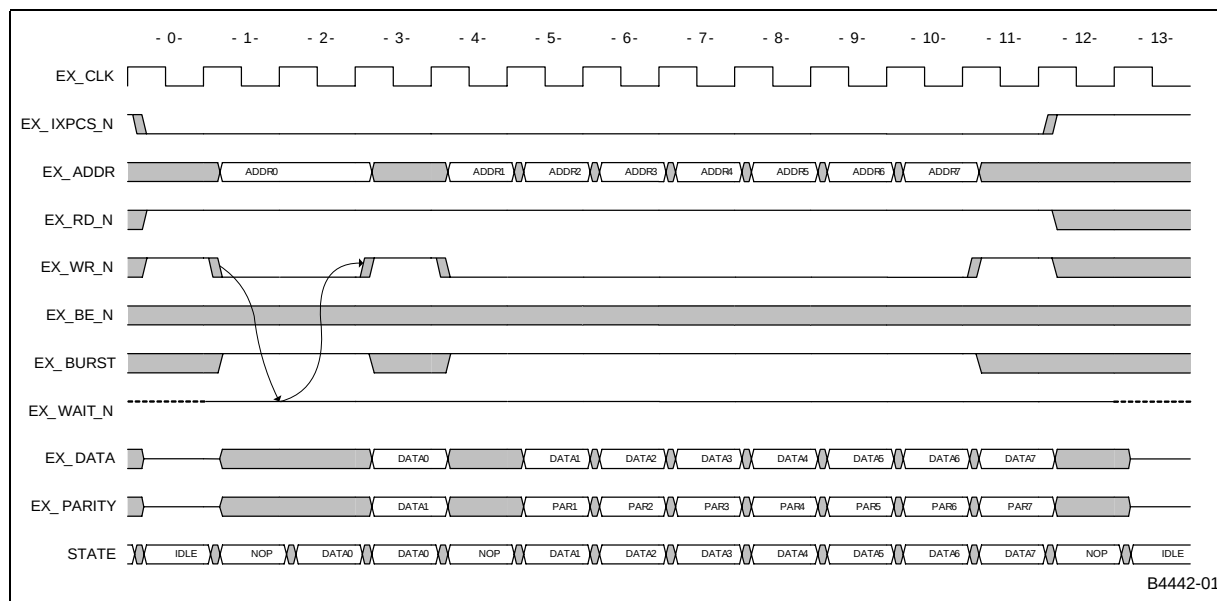


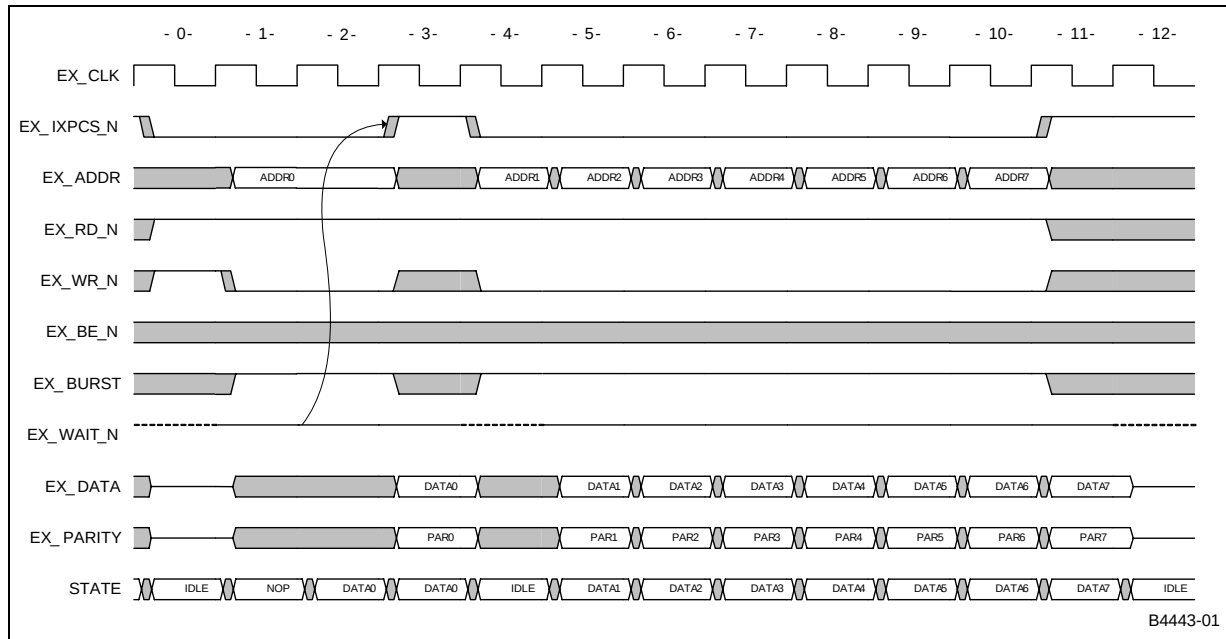
Figure 154 shows an external master choosing to insert NOP cycles at the start, in cycle 3, and in cycle 11 of a 8-word write. The external master can insert NOPS at anytime in the burst and the cycle will be ignored. Since the master started the burst with a NOP, EX_WAIT_N may have been asserted in cycle 2. In the above example, the Expansion bus controller did not assert EX_WAIT_N in cycle 2, therefore the external master



decided to start another NOP cycle in cycle 3. If EX_WAIT_N was asserted in cycle 2, the external master cannot start a NOP in cycle 3. It must wait until EX_WAIT_N is sampled deasserted.

12.4.5.6 Eight-Word Inbound Write with EX_SLAVE_CS_N Deassertion

Figure 155. Eight-Word Inbound Write with EX_SLAVE_CS_N Deassertion

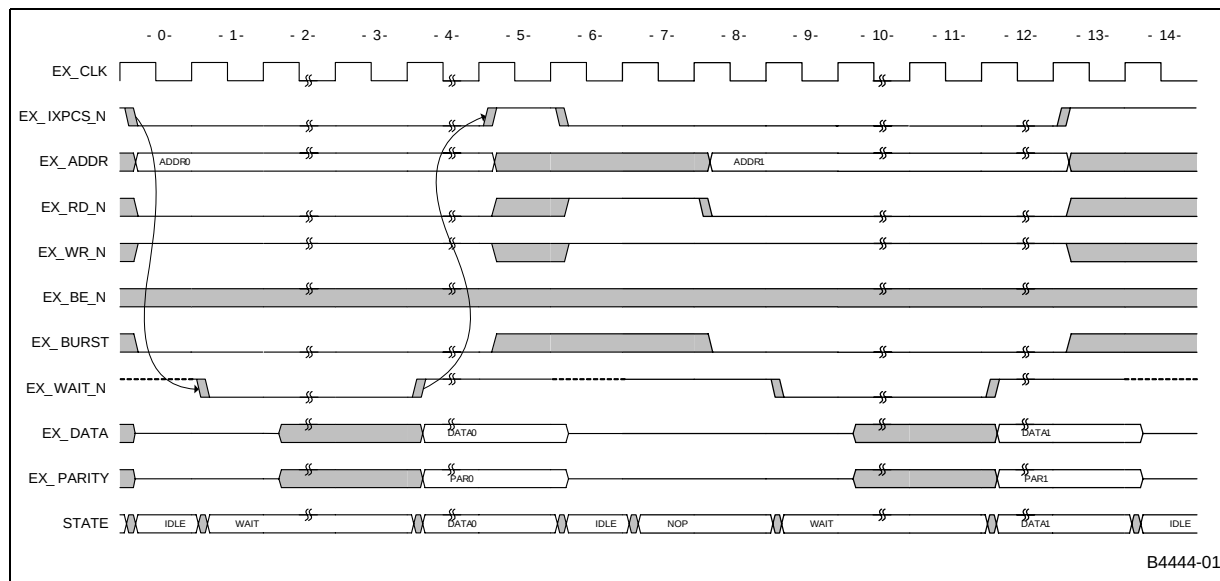


The above timing diagram shows an external master choosing to deassert EX_SLAVE_CS_N in the middle of an 8-word write. The external master can deassert EX_SLAVE_CS_N anytime after EX_WAIT_N is sampled deasserted. It can also deassert it between any of the 8 words being transferred. In the above diagram, the master deasserted EX_SLAVE_CS_N in cycle 3. When resuming the burst, the master must increment EX_ADDR[4:2] by 0x1. Once the transfer for EX_ADDR[4:2] = 0x7 is complete, the Expansion bus controller will transfer all 8 words to the AHB. The Expansion bus controller will never assert EX_WAIT_N on word 2-8 of the burst.



12.4.5.7 Back-to-Back 1-Word Inbound Reads with EX_SLAVE_CS_N

Figure 156. Back-to-Back 1-Word Inbound Reads with EX_SLAVE_CS_N

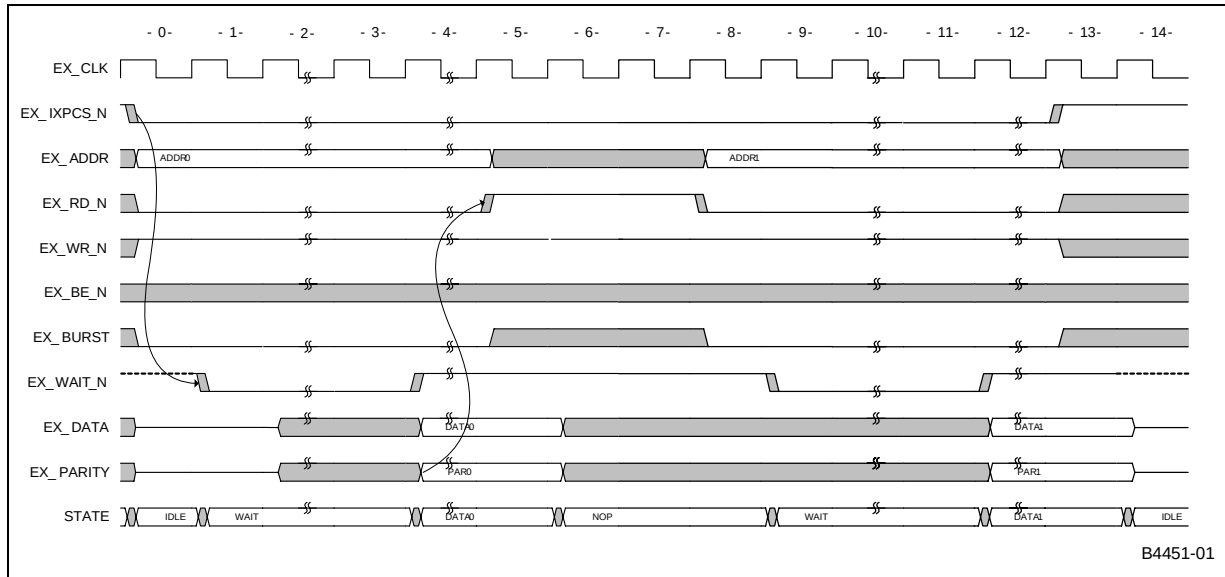


The above timing diagram shows two back-to-back 1-word read data transfers. For reads, the Expansion bus controller will always assert EX_WAIT_N one cycle after the assertion of EX_SLAVE_CS_N. As soon as the Expansion bus controller has the data it will deassert EX_WAIT_N. The deassertion of EX_WAIT_N is dependent on system activity and could be hundreds of cycles. After the master detects EX_WAIT_N deasserted, it can deassert EX_SLAVE_CS_N. However, the master can choose to extend cycle-4 and cycle-12 for as many cycles as it needs. The Expansion bus controller will continue to drive the read data until the end of the data transfer. In the second data transfer, the master chose to introduce 2 NOP cycles. The Expansion bus controller must ignore EX_ADDR during NOP cycles.



12.4.5.8 Back-to-Back 1-Word Reads without EX_SLAVE_CS_N Deasserted

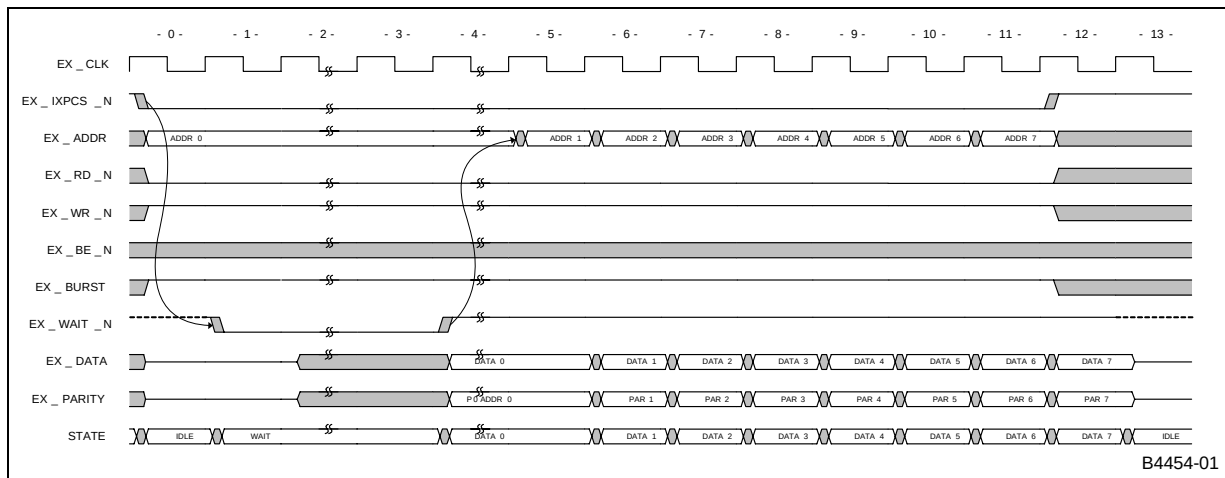
Figure 157. Back-to-Back 1-Word Reads without EX_SLAVE_CS_N Deasserted



The above timing diagram shows back-to-back 1-word reads with EX_SLAVE_CS_N not deasserted. In cycle-4, the master can choose to extend this cycle for as many cycles as it needs. The Expansion bus controller will re-assert EX_WAIT_N in cycle 9 when the 2nd read has started.

12.4.5.9 Eight-Word Inbound Read

Figure 158. Eight-Word Inbound Read

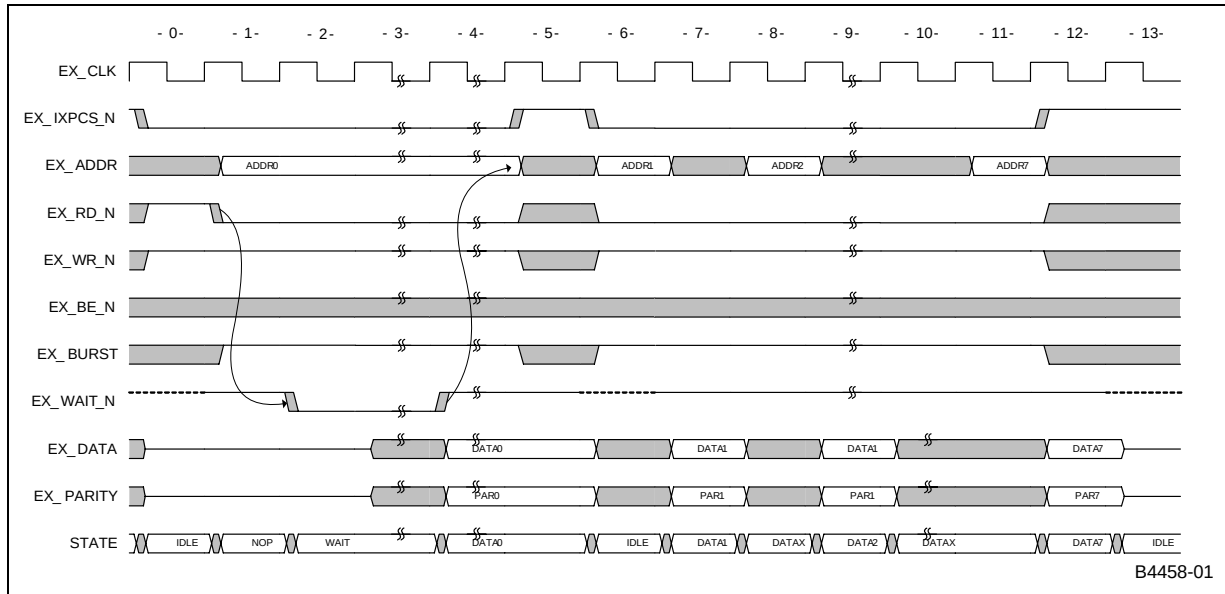


The above timing diagram shows an 8-word read data transfer. As soon as the Expansion bus controller deasserts EX_WAIT_N, the master can increment EX_ADDR, however cycle-4 can be extended as long as the master needs. The master can also stop the data burst in any cycle after EX_WAIT_N is deasserted.



12.4.5.11 Eight-Word Inbound Read with Deassertion of EX_SLAVE_CS_N

Figure 160. Eight-Word Inbound Read with Deassertion of EX_SLAVE_CS_N

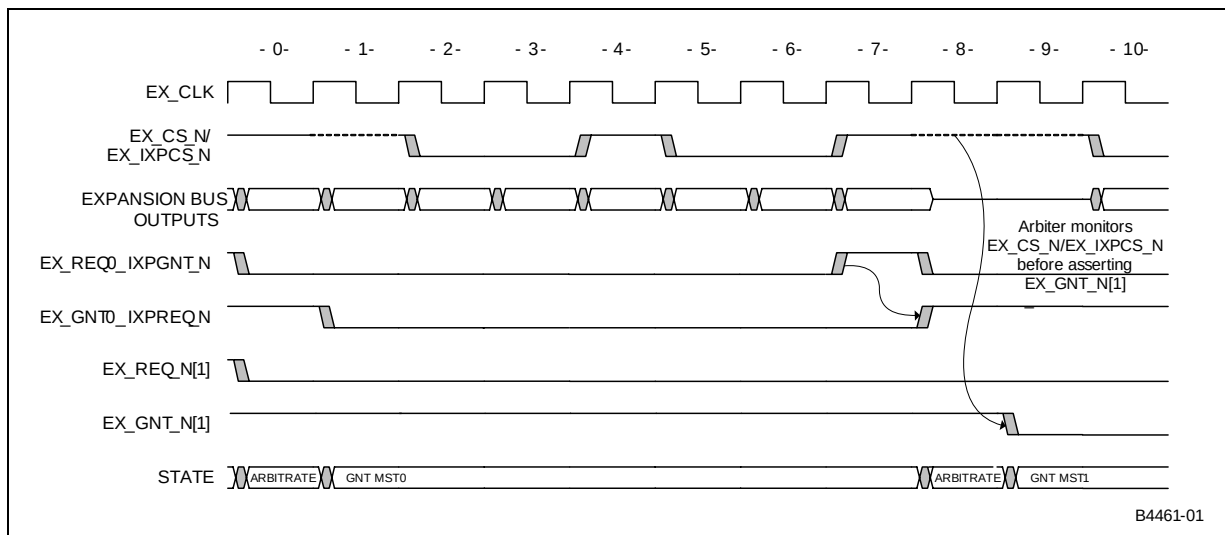


The above timing diagram shows an external master choosing to deassert EX_SLAVE_CS_N after the 1st word is transferred. The external master resumes the burst in cycle 6 and the Expansion bus controller does not assert EX_WAIT_N, since EX_ADDR[4:2] does not equal 0x0.

12.4.6 Expansion Bus Arbiter Timing Diagrams

12.4.6.1 Arbitration When GrantRemove Bit In EXP_MST_CONTROL is Set

Figure 161. Arbitration When GrantRemove Bit In EXP_MST_CONTROL is Set

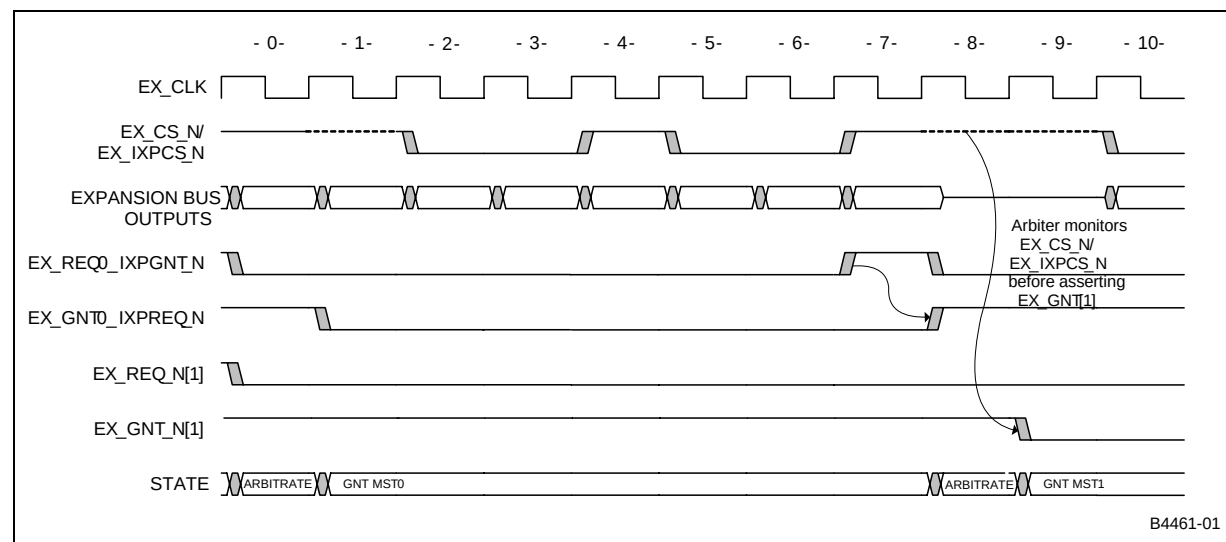




The above timing diagram shows the request and grant lines for two external masters. The Expansion bus controller owns the bus in cycles 0 and 1. The Expansion controller must tri-state the shared Expansion bus outputs on or before cycle 3. Master 0 owns the bus in cycles 3-5 and Master 1 owns the bus in cycles 7-10. In the above example, master 0 asserts EX_CS_N one cycle after its grant is asserted. Master 1 does not assert EX_CS_N until two cycles after its grant is asserted. A master can only assert EX_CS_N/EX_SLAVE_CS_N if its grant was asserted in the previous cycle. In this example, if master 1 did not assert EX_CS_N/EX_SLAVE_CS_N in cycle 9 it cannot assert it in cycle 10, since EX_GNT_N[1] was not asserted in cycle 9. A master may lose its grant in any cycle when GrantRemove = 1, however grant is always asserted for at least two cycles. When a master loses grant, it still owns the bus until the current transfer is complete. The Expansion bus controller monitors EX_CS_N and EX_SLAVE_CS_N to determine when the transfer is complete. The cycle after all the chip selects are disabled, the master must tri-state the bus. In this example master 0 must tri-state the shared Expansion bus pins in cycle 7.

12.4.6.2 Arbitration When GrantRemove Bit in EXP_MST_CONTROL is Clear

Figure 162. Arbitration When GrantRemove Bit in EXP_MST_CONTROL is Clear



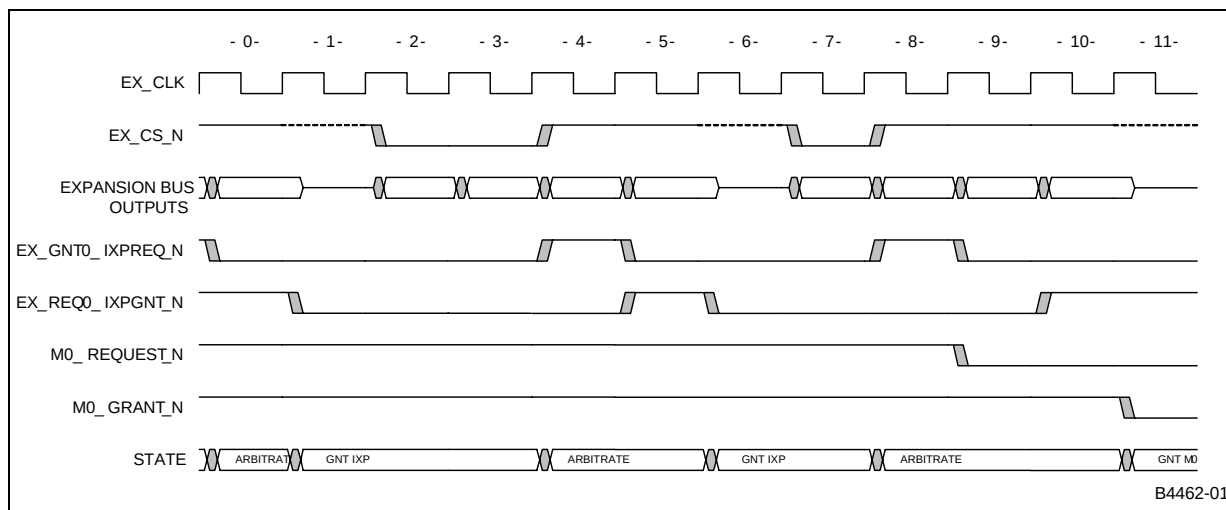
The above timing diagram shows the arbitration protocol when the GrantRemove bit is clear. Once grant is asserted to the master, that master will not lose grant until one cycle after its request is deasserted. In the above example, master 0 could start a new data transfer in cycle 8, however it chose not to start a new transfer. If a new transfer was started, the arbiter must not assert master 1's grant in cycle 9 and it must arbitrate after master 0 is finished with that transfer (and its request is deasserted). If master 0 started a new transfer in cycle 8, that master may not own the bus after that transfer completes. In this example master 0 tri-states the Expansion bus in cycle 8. Typically an external master will not start a new transaction in cycle 8, since it will complicate that master's grant logic.



12.4.7 External Expansion Bus Timing Diagram

12.4.7.1 External Arbiter Timing Diagram

Figure 163. External Arbiter Timing Diagram



The above timing diagram shows the Expansion bus controller requesting the bus in cycle 0. The external arbiter chooses to grant the Expansion bus controller in cycle 1. The Expansion bus controller owns the Expansion bus until it deasserts its request in cycle 4. The Expansion bus controller cannot start a new transfer in cycle 4 or 5 since it deasserted its request. In cycle 5, the Expansion bus controller asserts request, but grant is also lost. Since grant was lost in cycle 5, the Expansion bus controller tri-states the Expansion bus in cycle 6. The arbiter grants the Expansion bus controller in cycle 6 and the Expansion bus controller now owns the bus until it deasserts request in cycle 8. In cycle 9, the arbiter chose to park the bus on Expansion bus controller, however the arbiter chose to deassert grant in cycle 10. When bus parking occurs, the Expansion bus controller cannot start a new transaction until two cycles after its request is asserted and only if grant remains asserted. If the external arbiter chose to leave grant asserted to the Expansion bus controller in cycle 10, the Expansion bus controller would own the bus until it deasserts request again. When Expansion bus controller is parked on the bus, all shared Expansion bus outputs are driven to the deasserted state.

12.4.8 Configuration Straps

The Expansion bus controller contains configuration registers beyond what is required for its own configuration. There are several bits of configuration signals provided as output from the Expansion bus controller to the rest of the IXP45X/IXP46X network processors. These signals provide the AHB with functions like the software interrupt capabilities, location of Expansion bus controller in the processor memory map, PCI Host/Arbiter information, and configuration information on devices connected to the Expansion Bus.

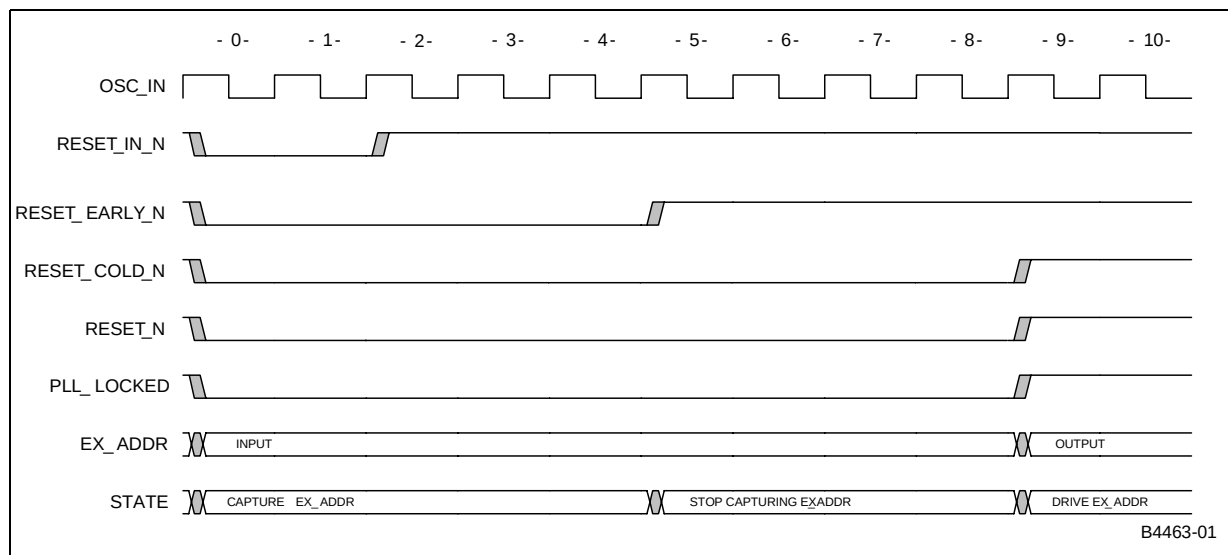
One of these general-purpose configuration registers is used to capture the value on the EX_ADDR pins immediately after reset. In the Expansion Bus Interface, 25 address lines are used to capture this configuration information during reset. When power up is complete and reset is asserted, the 25 address lines are configured as inputs. When internal reset is asserted (PLL_LOCK = 0), the configuration registers latch the values contained on the 25 address lines and the 25 address lines become configured as outputs. The Expansion Bus address signals have internal pull-up resistors of about



50Kohm that are only enabled when PLL_LOCK is deasserted. Pull-down resistors may be placed on some of the 25 address lines in order to select the configuration values at reset. The pull-down resistors must be able to over-drive the internal pull-up resistors. Please refer to [Section 12.5.9, “Configuration Register 0” on page 706](#) for additional details on particular configuration options. The Expansion bus controller samples EX_ADDR when reset_early_n = '0', which occurs during power up or during a watch dog reset. However, the Expansion bus controller only samples the Intel XScale processor Clock Set bits, EX_ADDR[23:21], when reset_early_n = '0' and reset_cold_n = '0'.

12.4.8.1 Sampling EX_ADDR During Reset

Figure 164. Sampling EX_ADDR During Reset



12.4.8.2 Expansion Bus Controller Operation

In this document, there are several occurrences of the word “must” in the description of the functional operation of the Expansion Bus Controller. Failure to comply to these requirements will result in unpredictable behavior of the Expansion Bus Controller.

12.5 Detailed Register Descriptions

Accesses to Expansion bus registers is only word transfers. Byte and half-word accesses result in unpredictable operation. Accesses to all reserved bits must be written with zero unless otherwise specified. All reserved address spaces must not be read. All writes to reserved address spaces results in unpredictable operation. External master sub-word accesses to the EXP_INBOUND_ADDR and EXP_LOCK[1:0] registers will be treated as a word access.

Table 225. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only



Table 225. Register Legend

Attribute	Legend	Attribute	Legend
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 226. Legacy Expansion Bus Register Summary

Address	Access	Name	Description
0xC4000000	RW	EXP_TIMING_CS0	Timing and Control Register for Chip Select 0
0xC4000004	RW	EXP_TIMING_CS1	Timing and Control Register for Chip Select 1
0xC4000008	RW	EXP_TIMING_CS2	Timing and Control Register for Chip Select 2
0xC400000C	RW	EXP_TIMING_CS3	Timing and Control Register for Chip Select 3
0xC4000010	RW	EXP_TIMING_CS4	Timing and Control Register for Chip Select 4
0xC4000014	RW	EXP_TIMING_CS5	Timing and Control Register for Chip Select 5
0xC4000018	RW	EXP_TIMING_CS6	Timing and Control Register for Chip Select 6
0xC400001C	RW	EXP_TIMING_CS7	Timing and Control Register for Chip Select 7
0xC4000020	RW	EXP_CNFG0	General-Purpose Configuration Register 0
0xC4000024	RW	EXP_CNFG1	General-Purpose Configuration Register 1

Table 227. Non-Legacy Expansion Bus Register Summary

Address	Register Name	Description	Reset Value	Attribute
0xC4000028	EXP_UNIT_FUSE_RESET	Specifies the value of the fuse register.	0xFFFFFFFF	See Register Table
0xC400002C	EXP_SMIIDLL	DLL bits for SMI used by the SMI DLL.	0x00000000	See Register Table
0xC4000100	EXP_MST_CONTROL	Specifies values for bus arbitration priority, bus master locking, and external master parity support	0x00000000	See Register Table
0xC4000104 and from External Master (See note below)	EXP_INBOUND_ADDR	Specifies the upper AHB address for inbound transfers.	0x00000008	See Register Table
0xC4000108 and from External Master (See note below)	EXP_LOCK0	This register is intended to facilitate resource locking for multiple masters	0x00000000	See Register Table
0xC400010C and from External Master (See note below)	EXP_LOCK1	This register is intended to facilitate resource locking for multiple masters	0x00000000	See Register Table
0xC4000120	EXP_PARITY_STATUS	Specifies the parity error status.	0x00000000	See Register Table
0xC4000124	EXP_SYNCINTEL_COUNT	This register is used to set the read latency count when a Synchronous Intel Device is accessed.	0x00000000	See Register Table



12.5.1 Timing and Control Registers for Chip Select 0

Register Name:				EXP_TIMING_CS0																												
Hex Offset Address:				0XC4000000								Reset Hex Value:				CS0: 0xBFFF3C4x																
Register Description:				Timing and Control Registers																												
Access: Read/Write																																
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13					9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3				T4		T5				CYCLE_ TYPE	CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN		

Note: The undefined (X) in the reset value is dependent upon values supplied to the chip on the Expansion Bus address during the reset sequence. Please refer to [Section 12.5.9](#), "Configuration Register 0" on page 706 for additional details.

12.5.2 Timing and Control Registers for Chip Select 1

Register Name:		EXP_TIMING_CS1																																					
Hex Offset Address:		0XC4000004								Reset Hex Value:				CS1: 0x00000000																									
Register Description:		Timing and Control Registers																																					
Access: Read/Write																																							
31	30	29	28	27	26	25				22	21	20	19				16	15	14	13					9	8	7	6	5	4	3	2	1	0					
CSx_EN	PAR_EN	T1		T2		T3				T4		T5				CYCLE_ TYPE		CNFG[4:0]				Sync_Intel		EXP_CHIP		BYTE_RD16		HRDY_POL		MUX_EN		SPLT_EN		WORD_EN		WR_EN		BYTE_EN	

12.5.3 Timing and Control Registers for Chip Select 2

Register Name:				EXP_TIMING_CS2																														
Hex Offset Address:				0XC4000008								Reset Hex Value:				CS2: 0x00000000																		
Register Description:				Timing and Control Registers																														
Access: Read/Write.																																		
31	30	29	28	27	26	25				22	21	20	19				16	15	14	13					9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3				T4		T5				CYCLE_ TYPE		CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN			



12.5.4 Timing and Control Registers for Chip Select 3

Register Name:		EXP_TIMING_CS3																													
Hex Offset Address:		0XC400000C								Reset Hex Value:				CS3: 0x00000000																	
Register Description:		Timing and Control Registers																													
Access: Read/Write																															
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3			T4		T5			CYCLE_ TYPE		CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN		

12.5.5 Timing and Control Registers for Chip Select 4

Register Name:		EXP_TIMING_CS4																													
Hex Offset Address:		0XC4000010								Reset Hex Value:				CS4: 0x00000000																	
Register Description:		Timing and Control Registers																													
Access: Read/Write.																															
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3			T4		T5			CYCLE_ TYPE	CNFG[4:0]					Sync_Intel	EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN			

12.5.6 Timing and Control Registers for Chip Select 5

Register Name:				EXP_TIMING_CS5																											
Hex Offset Address:				0XC4000014								Reset Hex Value:				CS5: 0x00000000															
Register Description:				Timing and Control Registers																											
Access: Read/Write.																															
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3			T4		T5			CYCLE_TYPE		CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN		



12.5.7 Timing and Control Registers for Chip Select 6

Register Name:				EXP_TIMING_CS6																											
Hex Offset Address:				0XC4000018								Reset Hex Value:				CS6: 0x00000000															
Register Description:				Timing and Control Registers																											
Access: Read/Write.																															
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3				T4		T5				CYCLE_ TYPE	CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN	

12.5.8 Timing and Control Registers for Chip Select 7

Register Name:		EXP_TIMING_CS7																													
Hex Offset Address:		0XC400001C								Reset Hex Value:		CS7: 0x00000000																			
Register Description:		Timing and Control Registers																													
Access: Read/Write.																															
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR_EN	T1		T2		T3				T4		T5				CYCLE_ TYPE	CNFG[4:0]				Sync_Intel		EXP_CHIP	BYTE_RD16	HRDY_POL	MUX_EN	SPLT_EN	WORD_EN	WR_EN	BYTE_EN	

Table 228. Bit Level Definition for each of the Timing and Control Registers (Sheet 1 of 2)

Bits	Name	Description
31	CSx_EN	0 = Chip Select x disabled 1 = Chip Select x enabled
30	PAR_EN	0 = Parity is not generated or compared 1 = Parity is generated and compared Parity is only supported for Intel, Motorola, and Micron ZBT modes.
29:28	T1 – Address timing	00 = Generate normal address phase timing 01 - 11 = Extend address phase by 1 - 3 clocks
27:26	T2 – Setup / Chip Select Timing	00 = Generate normal setup phase timing 01 - 11 = Extend setup phase by 1 - 3 clocks
25:22	T3 – Strobe Timing	0000 = Generate normal strobe phase timing 0001-1111 = Extend strobe phase by 1 - 15 clocks
21:20	T4 – Hold Timing	00 = Generate normal hold phase timing 01 - 11 = Extend hold phase by 1 - 3 clocks
19:16	T5 – Recovery Timing	0000 = Generate normal recovery phase timing 0001-1111 = Extend recovery phase by 1 - 15 clocks



Table 228. Bit Level Definition for each of the Timing and Control Registers (Sheet 2 of 2)

Bits	Name	Description
15:14	CYC_TYPE	00 = Configures the Expansion bus for Intel cycles. 01 = Configures the Expansion bus for Motorola cycles. 10 = Configures the Expansion bus for HPI cycles. (HPI reserved for chip selects [7:4] only) 11 = Configures the Expansion bus for Micron ZBT cycles
13:9	CNFG[4:0]	Device Configuration Size. Calculated using the formula: SIZE OF ADDR SPACE = $2^{(9+CNFG[4:1]+16*CNFG[0])}$ For Example: 00000 = Address space of 2^9 = 512 Bytes 00010 = Address space of 2^{10} = 1024 Bytes ... 10000 = Address space of 2^{17} = 128 Kbytes ... 11100 = Address space of 2^{23} = 8 Mbytes 11110 = Address space of 2^{24} = 16 Mbytes 00001 = Address space of 2^{25} = 32Mbytes
8	Sync_Intel	Synchronous Intel StrataFlash® select. This bit must be 0 if CYC_TYPE is not programmed to Intel cycles. 0 = Target device is not a Synchronous Intel StrataFlash 1 = Target device is a Synchronous Intel StrataFlash
7	EXP_CHIP	0 = Target device is not one of the IXP45X/IXP46X network processors 1 = Target device is one of the IXP45X/IXP46X network processors. This bit must only be set to 1 when CYC_TYPE is configured to be Intel Cycles and Sync_Intel is set to 0.
6	BYTE_RD16	Byte read access to Half Word device 0 = Byte access disabled. 1 = Byte access enabled.
5	HRDY_POL	HPI HRDY polarity (reserved for EX_CS_N[7:4] only) 0 = Polarity low true. 1 = Polarity high true.
4	MUX_EN	0 = Separate address and data buses. 1 = Multiplexed address / data on data bus.
3	SPLT_EN	0 = AHB split transfers disabled. 1 = AHB split transfers enabled.
2	WORD_EN	1 = Expansion bus uses 32-bit data bus 0 = Expansion bus uses 8/16 bit data bus based on BYTE_EN bit
1	WR_EN	0 = Writes to CS region are disabled. 1 = Writes to CS region are enabled.
0	BYTE_EN	0 = Expansion bus uses 16-bit-wide data bus if WORD_EN = 0. 1 = Expansion bus uses only 8-bit data bus if WORD_EN = 0.

The EXP_TIMING_CS registers may only be written if there is not an outstanding Expansion bus transaction. Software must ensure that all outstanding Expansion bus transfers are complete before changing the EXP_TIMING_CS registers.

12.5.9 Configuration Register 0

At power up or whenever RESET_IN_N is asserted, the expansion bus address outputs are switched to inputs and the states of the bits are captured and stored in Configuration Register 0, bits 23 through 0. This occurs when PLL_LOCK is deasserted.



These configuration bits are made available to the system as outputs from the Expansion bus controller block. With the exception of bits 23, 22 and 21, which are read only, all other bits may be written and read from the AHB. However software should only modify the value of MEM_MAP (bit 31) and leave all other values as the captured value.

Register Name:		EXP_CNFG0																																
Hex Offset Address:		0XC4000020								Reset Hex Value:		0x80XXXXXX																						
Register Description:		Configuration Register #0																																
Access: Read/Write.																																		
31	30						24	23	22	21	20				17	16	15						10	9	8	7	6	5	4	3	2	1	0	
MEM_MAP	(Reserved)							Clock Set				Customer				(Reserved)							IOWAIT_CS0	EXP_MED_DRIVE	USB CLOCK	32_FLASH	EXP_ARB	EXP_DRIVE	PCI_CLK			PCI_ARB	PCI_HOST	8/16

Table 229. Configuration Register 0 Description (Sheet 1 of 2)

Bit	Name	Reset Value	Description
31	MEM_MAP	1	Location of Expansion Bus in memory map space: 0 = Located at "50000000" (normal mode) 1 = Located at "00000000" (boot mode)
30:24	(Reserved)	0x0	(Reserved)
23:21	Intel XScale® Processor Clock Set[2:0]	EX_ADDR[23:21]	Allow a slower Intel XScale processor clock speed to override device fuse settings. However cannot be used to over clock core speed. Refer to Table 230, "Setting The Intel XScale® Processor Operation Speed" on page 709 for additional details.
20:17	Customer	EX_ADDR[20:17]	Customer defined bits.
16:11	(Reserved)	EX_ADDR[16:11]	(Reserved)
10	IOWAIT_CS0	EX_ADDR[10]	1 = EX_IOWAIT_N is sampled during the read/write Expansion bus cycles as defined in Section 12.4.1.5, "Using I/O Wait" on page 661 for Chip Select 0. 0 = EX_IOWAIT_N is ignored for read and write cycles to Chip select 0 if EXP_TIMING_CS0 is configured to Intel mode. Typically, IOWAIT_CS0 must be pulled down to Vss when attaching a Synchronous Intel StrataFlash on Chip select 0 since the default mode for EXP_TIMING_CS0 is Intel mode and EX_IOWAIT_N is an unknown value for Synchronous Intel StrataFlash. If the board does not connect the Synchronous Intel StrataFlash WAIT pin to EX_WAIT_N (and the board guarantees EX_IOWAIT_N is pulled up), the value of IOWAIT_CS0 is a don't care since EX_IOWAIT_N will not be asserted. When EXP_TIMING_CS0 is reconfigured to Intel Synchronous mode during boot-up (for Synchronous Intel chips), the Expansion bus controller ignores EX_IOWAIT_N during read and write cycles since the WAIT functionality is determined from the EXP_SYNCINTEL_COUNT and EXP_TIMING_CS registers.
9	EXP_MEM_DRIVE	EX_ADDR[9]	Refer to the table found in EXP_DRIVE bit (bit 5) of this register.



Table 229. Configuration Register 0 Description (Sheet 2 of 2)

Bit	Name	Reset Value	Description
8	USB Clock	EX_ADDR[8]	Controls the USB clock select 1 = USB Host/Device clock is generated internally 0 = USB Device clock is generated from GPIO[0]. USB Host clock is generated from GPIO[1]. When generating a spread spectrum clock on OSC_IN, GPIO[0] can be driven from the system board to generate a 48 MHz clock for the USB Device and GPIO[1] can be driven from the system board to generate a 60 MHz clock for the USB Host.
7	32_FLASH	EX_ADDR[7]	Refer to the table found in 8/16_FLASH bit (bit 0) of this register.
6	EXP_ARB	EX_ADDR[6]	Configures the Expansion bus arbiter 0 = External arbiter for Expansion bus. 1 = Expansion bus controller arbiter enabled
5	EXP_DRIVE	EX_ADDR[5]	Expansion bus low/medium/high drive strength. The drive strength depends on EXP_DRIVE and EXP_MEM_DRIVE configuration bits. EXP_MEM_DRIVE EXP_DRIVE Expansion drive strength ----- 0 0 Reserved 0 1 Medium Drive 1 0 Low Drive 1 1 High Drive
4	PCI_CLK	EX_ADDR[4]	Sets the clock speed of the PCI Interface 0 = 33 MHz 1 = 66 MHz
3	(Reserved)	EX_ADDR[3]	(Reserved). EX_ADDR[3] must not be pulled down during address strapping. This bit must be written to '1' if performing a write to this register.
2	PCI_ARB	EX_ADDR[2]	Enables the PCI Controller Arbiter 0 = PCI arbiter disabled 1 = PCI arbiter enabled
1	PCI_HOST	EX_ADDR[1]	Configures the PCI Controller as PCI Bus Host 0 = PCI as non-host 1 = PCI as host
0	8/16_FLASH	EX_ADDR[0]	Specifies the data bus width of the FLASH memory device found on Chip Select 0. The data bus is based upon bits 0 and 7 of Configuration Register 0. 32_FLASH 8/16_FLASH Data bus size ----- 0 0 16-bit 0 1 8-bit 1 0 Reserved 1 1 32-bit

The chip-level memory map used is determined by the state of bit 31. At system reset this bit is a '1' and the memory map places the Expansion Bus at address 0x00000000 through 0x0FFFFFFF. This allows boot code stored in flash to be retrieved and executed as required.

Once the boot sequence completes this bit is written to a '0', switching the default system memory map to place the DDRI SDRAM controller at address 0x00000000 to 0x0FFFFFFF. The Expansion bus controller now resides at address 0x50000000 to 0x5FFFFFFF. Weak pull-up resistors are placed on each expansion bus address pin.



Table 230. Setting The Intel XScale® Processor Operation Speed

Intel XScale® Processor Speed (Factory Part Speed)	Cfg0 EX_ADDR[21]	Cfg1 EX_ADDR[22]	Cfg_en_n EX_ADDR[23]	Actual Core Speed (MHz)
667 MHz	X	X	1	667 MHz
667 MHz	0	0	0	667 MHz
667 MHz	1	0	0	533 MHz
667 MHz	0	1	0	266 MHz
667 MHz	1	1	0	400 MHz
533 MHz	X	X	1	533 MHz
533 MHz	0	0	0	533 MHz
533 MHz	1	0	0	533 MHz
533 MHz	0	1	0	266 MHz
533 MHz	1	1	0	400 MHz
400 MHz	X	X	1	400 MHz
400 MHz	0	0	0	400 MHz
400 MHz	1	0	0	400 MHz
400 MHz	0	1	0	266 MHz
400 MHz	1	1	0	400 MHz
266 MHz	X	X	X	266 MHz

Note that the Intel XScale processor can operate at slower speeds than the factory programmed speed setting. This is done by placing a value on Expansion bus address bits 23,22,21 when PLL_LOCK is deasserted and knowing the speed grade of the part from the factory. Column 1 above denotes the speed grade of the part from the factory. Column 2, 3, and 4 denotes the values captured on the Expansion Bus address bits when PLL_LOCK is deasserted. Column 5 represents the speed at which the Intel XScale processor speed will now be operating at.

12.5.10 Configuration Register 1

One additional configuration register is defined within the Expansion bus controller for use by the IXP45X/IXP46X network processors.



Register Name:		EXP_CNFG1																																																
Hex Offset Address:		0XC4000024								Reset Hex Value:		0x00000000																																						
Register Description:		Configuration Register #1																																																
Access: Read/Write.																																																		
31												20	19	18	17	16	15	14	13	12	11	10	9	8	7						2	1	0																	
MPI_EN	(Reserved)										SMII SOURCE SYNC				NPE-C SMII		NPE-B SMII		NPE-A SMII		(Reserved)		NPE-C ERR_EN		NPE-B ERR_EN		NPE-A ERR_EN		(Reserved)		EXP_BYTE_SWAP_EN		FORCE_BYTE_SWAP		BYTE_SWAP_EN		(Reserved)										SW_INT1		SW_INT0	

Table 231. Expansion Bus Configuration Register 1-Bit Definition (Sheet 1 of 3)

Bit	Name	Description
31	MPI_EN	0 = DDR transactions are routed through the AHB 1 = DDR transactions are routed through the MPI port. This bit should always be set when configuring the DDR during boot-up. When this bit is set, the performance between the Intel XScale® Processor and DDR is increased.
30:20	(Reserved)	(Reserved)
19	SMII SOURCE SYNC	0 = SMII mode for NPE-A, NPE-B, NPE-C does not operate in source synchronous mode 1 = Reserved This bit must be configured during boot-up. Set to 0 for the IXP45X/IXP46X network processors.
Note: For transactions initiated by the Intel XScale® Processor, the selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Intel® IXP4XX Product Line Memory Management Unit (MMU) and the BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit will be from Expansion bus controller Configuration Register 1, Bit 8. This bit will reset to 0. The default endian conversion method for IXP45X/IXP46X network processors is address coherency. This was selected to enable backward compatible with the Intel® IXP425 processor. The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit. When the bit is 0, address coherency is always performed. When the bit is 1, the type of coherency depends on the P-attribute bit. The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output, from the Intel XScale processor, with any store or load access associated with that page. Note: When enabling SMII mode for the NPE's during boot-up, the NPE's will need to be reset in software to ensure a clean transition into SMII mode. Software must execute the following code when entering SMII mode: 1. Write the Expansion EXP_UNIT_FUSE_RESET register to turn ON the reset for the NPE's. 2. If the PCI interface is not used, the PCI RCOMP bit of the EXP_UNIT_FUSE_RESET register must be set to a '1'. If the PCI interface is used, the PCI RCOMP bit must be left unchanged at logic '0'. 3. Read the SMII_RCOMP_CSR register, set bit 16 to '1', and write value back to SMII_RCOMP_CSR register. 4. Write the EXP_SMIIDLL register to enable the DLL. 5. Write the Expansion EXP_CNFG1 register to turn on SMII mode for the appropriate NPE's (SMII mode for NPE-B must be enabled for SMII mode to work on NPE-A or NPE-C) 6. Read the Expansion EXP_CNFG1 register to ensure previous write has occurred 7. Wait at least 1000 ns 8. Write the Expansion EXP_UNIT_FUSE_RESET register to turn OFF the reset for the NPE's. 9. Resume normal boot-up 10. Wait at least 12ms before starting Ethernet traffic on ports that have SMII enabled, since it takes 12 ms for the SMII pins to stabilize.		



Table 231. Expansion Bus Configuration Register 1-Bit Definition (Sheet 2 of 3)

Bit	Name	Description
18	NPE-C SMII	0 = MII mode enabled for NPE-C 1 = SMII mode enabled for NPE-C This bit must be configured during boot-up and cannot change unless a system reset is performed. Note: See note, below.
17	NPE-B SMII	0 = MII mode enabled for NPE-B 1 = SMII mode enabled for NPE-B Refer to Table 233 for more details. This bit must be configured during boot-up and cannot change unless a system reset is performed Note: See note, below.
16	NPE-A SMII	0 = MII mode enabled for NPE-A 1 = SMII mode enabled for NPE-A This bit must be configured during boot-up and cannot change unless a system reset is performed. Note: See note, below.
15	(Reserved)	(Reserved)
14	NPE-C ERR_EN	0 = Error handling in NPE-C is disabled 1 = Error handling in NPE-C is enabled. For more information, see "Error Handling" on page 951 .
13	NPE-B ERR_EN	0 = Error handling in NPE-B is disabled 1 = Error handling in NPE-B is enabled. For more information, see "Error Handling" on page 951 .
12	NPE-A ERR_EN	0 = Error handling in NPE-A is disabled 1 = Error handling in NPE-A is enabled. For more information, see "Error Handling" on page 951 .
11	(Reserved)	(Reserved)
Note: For transactions initiated by the Intel XScale® Processor, the selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Intel® IXP4XX Product Line Memory Management Unit (MMU) and the BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit will be from Expansion bus controller Configuration Register 1, Bit 8. This bit will reset to 0. The default endian conversion method for IXP45X/IXP46X network processors is address coherency. This was selected to enable backward compatible with the Intel® IXP425 processor. The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit. When the bit is 0, address coherency is always performed. When the bit is 1, the type of coherency depends on the P-attribute bit. The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output, from the Intel XScale processor, with any store or load access associated with that page. Note: When enabling SMII mode for the NPE's during boot-up, the NPE's will need to be reset in software to ensure a clean transition into SMII mode. Software must execute the following code when entering SMII mode: 1. Write the Expansion EXP_UNIT_FUSE_RESET register to turn ON the reset for the NPE's. 2. If the PCI interface is not used, the PCI RCOMP bit of the EXP_UNIT_FUSE_RESET register must be set to a '1'. If the PCI interface is used, the PCI RCOMP bit must be left unchanged at logic '0'. 3. Read the SMII_RCOMP_CSR register, set bit 16 to '1', and write value back to SMII_RCOMP_CSR register. 4. Write the EXP_SMIIDLL register to enable the DLL. 5. Write the Expansion EXP_CNFG1 register to turn on SMII mode for the appropriate NPE's (SMII mode for NPE-B must be enabled for SMII mode to work on NPE-A or NPE-C) 6. Read the Expansion EXP_CNFG1 register to ensure previous write has occurred 7. Wait at least 1000 ns 8. Write the Expansion EXP_UNIT_FUSE_RESET register to turn OFF the reset for the NPE's. 9. Resume normal boot-up 10. Wait at least 12ms before starting Ethernet traffic on ports that have SMII enabled, since it takes 12 ms for the SMII pins to stabilize.		



Table 231. Expansion Bus Configuration Register 1-Bit Definition (Sheet 3 of 3)

Bit	Name	Description
10	EXP_BYTE_SWAP_EN	Enables byte swapping for EXP inbound transfers. This swapping occurs regardless of the Intel XScale processor endianism. 0 = byte swapping disabled 1 = byte swapping enabled
9	FORCE_BYTE_SWAP	Forces byte swapping on Intel XScale processor initiated accesses in little-endian mode regardless of the P-attribute bit. 0 = do not force byte swapping 1 = force byte swapping on all transfers regardless of the P-attribute bit in little-endian mode. When this bit is set, BYTE_SWAP_EN is ignored and byte swapping always happens in little-endian mode.
8	BYTE_SWAP_EN	Sets byte swapping at the on Intel XScale processor initiated accesses 0 = byte swapping disabled 1 = byte swapping enabled Note: See note, below.
7:2	(Reserved)	(Reserved)
1	SW_INT1	0 = Disable interrupt 1 = Generate interrupt
0	SW_INT0	0 = Disable interrupt 1 = Generate interrupt
Note: For transactions initiated by the Intel XScale® Processor, the selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Intel® IXP4XX Product Line Memory Management Unit (MMU) and the BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit will be from Expansion bus controller Configuration Register 1, Bit 8. This bit will reset to 0. The default endian conversion method for IXP45X/IXP46X network processors is address coherency. This was selected to enable backward compatible with the Intel® IXP425 processor. The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit. When the bit is 0, address coherency is always performed. When the bit is 1, the type of coherency depends on the P-attribute bit. The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output, from the Intel XScale processor, with any store or load access associated with that page. Note: When enabling SMII mode for the NPE's during boot-up, the NPE's will need to be reset in software to ensure a clean transition into SMII mode. Software must execute the following code when entering SMII mode: 1. Write the Expansion EXP_UNIT_FUSE_RESET register to turn ON the reset for the NPE's. 2. If the PCI interface is not used, the PCI RCOMP bit of the EXP_UNIT_FUSE_RESET register must be set to a '1'. If the PCI interface is used, the PCI RCOMP bit must be left unchanged at logic '0'. 3. Read the SMII_RCOMP_CSR register, set bit 16 to '1', and write value back to SMII_RCOMP_CSR register. 4. Write the EXP_SMIIDLL register to enable the DLL. 5. Write the Expansion EXP_CNFG1 register to turn on SMII mode for the appropriate NPE's (SMII mode for NPE-B must be enabled for SMII mode to work on NPE-A or NPE-C) 6. Read the Expansion EXP_CNFG1 register to ensure previous write has occurred 7. Wait at least 1000 ns 8. Write the Expansion EXP_UNIT_FUSE_RESET register to turn OFF the reset for the NPE's. 9. Resume normal boot-up 10. Wait at least 12ms before starting Ethernet traffic on ports that have SMII enabled, since it takes 12 ms for the SMII pins to stabilize.		

Under software control, bits 0 and 1 allow interrupts to be generated to the Interrupt Controller.

When byte swapping is enabled, byte 3 of a word is swapped with byte 0, and byte 1 is swapped with byte 2.

Register Name:							EXP_UNIT_FUSE_RESET																								
Physical Address:							0xC4000028				Reset Hex Value:				0XXXXXXXXX																
Register Description:							Specifies the value of the fuse register.																								
Access: See below.																															
31							24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							SPEED	RSA	NPE-B ETH 1-3	NPE-A ETH	USB HOST	UTOPIA PHY	RESERVED	PCI	NPE-C	NPE-B	NPE-A	ETHERNET 1	ETHERNET 0	UTOPIA	HSS	RESERVED	HDLC	DES	AES	HASHING COP	USB DEVICE	RESERVED			

Register		EXP_UNIT_FUSE_RESET		
Bits	Name	Description	Reset Value	Access
31:2 3	Reserved	Reserved	0x0	RO
23:2 2	SPEED	Intel XScale® Processor FACTORY SPEED 00 = 533 MHz 01 = 400 MHz 10 = 667 MHz 11 = 266 MHz	FUSE[7:6]	R0
21	RSA	0 = RSA Crypto Block Enabled 1 = RSA Crypto Block Disabled and clock gated	FUSE[5]	RW
20	NPE-B ETHERNET 1-3	0 = NPE-B 1-3 Ethernet Enabled 1 = NPE-B 1-3 Ethernet Disabled and clock gated Refer to Table 233 for more details.	FUSE[4]	RW
19	NPE-A ETHERNET	0 = NPE-A Ethernet Enabled if UTOPIA=1 1 = NPE-A Ethernet Disabled Refer to Table 232 for more details.	FUSE[3]	RW
18	USB HOST	0 = USB HOST Enabled 1 = USB HOST Disabled and clock gated	FUSE[2]	RW
17:1 6	UTOPIA Phy	00 = Maximum of 31 UTOPIA Phys 01 = Maximum of 16 UTOPIA Phys 10 = Maximum of 8 UTOPIA Phys 11 = Maximum of 4 UTOPIA Phys	FUSE[1:0]	RW
15	ECC/1588 Timesync	0 = DDR ECC and 1588 Timesync Enabled 1 = DDR ECC and 1588 Timesync Disabled	FUSE[23]	RW
14	PCI	0 = PCI Enabled 1 = PCI Disabled and clock gated	FUSE[22]	RW
13	NPE-C	0 = NPE-C Enabled 1 = NPE-C Disabled and clock gated/Reset	FUSE[21]	RW
12	NPE-B	0 = NPE-B Enabled 1 = NPE-B Disabled and clock gated/Reset	FUSE[20]	RW
11	NPE-A	0 = NPE-A Enabled 1 = NPE-A Disabled and clock gated/Reset	FUSE[19]	RW



Register		EXP_UNIT_FUSE_RESET		
Bits	Name	Description	Reset Value	Access
10	NPE-C ETHERNET	0 = NPE-C Ethernet Enabled 1 = NPE-C Ethernet Disabled	FUSE[18]	RW
9	NPE-B ETHERNET 0	0 = NPE-B Ethernet 0 Enabled 1 = NPE-B Ethernet 0 Disabled Refer to Table 233 for more details.	FUSE[17]	RW
8	UTOPIA	0 = UTOPIA Enabled 1 = UTOPIA Disabled/NPE-A ethernet enabled if NPE-A ETHERNET=0 Refer to Table 232 for more details.	FUSE[16]	RW
7	HSS	0 = HSS Enabled 1 = HSS Disabled and clock gated	FUSE[13]	RW
6	Reserved	Reserved	FUSE[14]	RW
5	HDLC	0 = HDLC Enabled 1 = HDLC Disabled and clock gated	FUSE[13]	RW
4	DES	0 = DES Enabled 1 = DES Disabled	FUSE[10]	RW
3	AES	0 = AES Enabled 1 = AES Disabled	FUSE[10]	RW
2	Hashing Cop	0 = HASH Enabled 1 = HASH Disabled	FUSE[10]	RW
1	USB Device	0 = USB DEVICE Enabled 1 = USB DEVICE Disabled and clock gated	FUSE[9]	RW
0	PCI RCOMP	0 = PCI RCOMP Enabled 1 = PCI RCOMP Disabled This bit must be set to a '1' if the PCI interface is not being used and SMII mode is enabled.	0	RW

The EXP_UNIT_FUSE_RESET register can be read by software to determine which features are fused out or being software disabled. Software can choose to disable features for blocks that are not being used to lower power consumption by writing a '1' to the appropriate bit for the feature that needs to be disabled. Disabling features by software is recommended to be done during the system initialization phase (i.e. before the start or use of the related feature to be disabled). A feature cannot be enabled after being disabled without asserting system reset. Any block can be independently disabled by software, however the HDLC and HSS must always be both enabled or both disabled by software since these blocks work together. For blocks that have external pin interfaces, pull-ups or pull-downs are enabled in the PAD I/O, and these pins should be left unconnected in the system.

Each of the NPE's can also be reset anytime through software by writing a '1' to the appropriate NPE bit (NPE-A, NPE-B, or NPE-C) in the EXP_UNIT_FUSE_RESET register. When a NPE is reset, all the coprocessors that belong to that NPE are also automatically reset and all the coprocessor data transfers to external interfaces are terminated immediately. When resetting the NPE's, software should do a read modify write to the EXP_UNIT_FUSE_RESET register and only set the NPE bit that needs to be reset. A read modify write must ensure that the NPE coprocessors that were previously disabled continue to be disabled and the coprocessors that are enabled continue to be enabled. Writing a '1' to a NPE coprocessor bit that is already enabled when resetting the NPE's will disable that coprocessor and cause the coprocessor interface pins to enter an unknown state. When software writes a '1' to reset a NPE, the EXP_UNIT_FUSE_RESET register will not reflect the reset status until that NPE has been reset. Typically software will poll the EXP_UNIT_FUSE_RESET register until that NPE bit that is being reset



equals a '1'. Once software detects that the NPE has been reset, software then can clear the appropriate bits of the EXP_UNIT_FUSE_RESET register to deassert reset to that NPE.

When software does a NPE reset, the coprocessor interfaces that are attached to that NPE could enter reset at any time, even during the middle of a data transfer. The system design must account for this abrupt reset on these busses. After being reset, the coprocessor interfaces will go back to the default reset mode for that interface.

Additionally, the Intel XScale processor factory speed cannot be changed by software.

Table 232. UTOPIA/Ethernet Configuration Options

UTOPIA Fuse	NPE-A Ethernet Fuse	NPE-A SMII Config Bit	Operation
0	0	0	UTOPIA interface enabled on NPE-A. NPE-A ethernet interface is disabled. When the UTOPIA interface is enabled, the NPE-A SMII config bit must be 0.
0	1	0	UTOPIA interface enabled on NPE-A. NPE-A ethernet interface is disabled. When the UTOPIA interface is enabled, the NPE-A SMII config bit must be 0.
1	0	0	UTOPIA interface is disabled. NPE-A ethernet interface is enabled in MII mode.
1	0	1	UTOPIA interface is disabled. NPE-A ethernet interface is enabled in SMII mode.
1	1	X	UTOPIA interface is disabled. NPE-A ethernet interface is disabled.

Table 233. NPE-B Ethernet Configuration Options

NPE-B Ethernet 0 Fuse	NPE-B Ethernet 1-3 Fuse	NPE-B SMII Config Bit	Operation
0	0	0	NPE-B Ethernet 0 is enabled in MII mode. NPE-B Ethernet 1-3 is disabled.
0	1	0	NPE-B Ethernet 0 is enabled in MII mode. NPE-B Ethernet 1-3 is disabled and clock gated.
1	0	X	Illegal mode and will result in unpredictable operation.
1	1	X	NPE-B Ethernet 0 is disabled. NPE-B Ethernet 1-3 is disabled and clock gated.
0	0	1	NPE-B Ethernet 0 and Ethernet 1-3 are enabled in SMII mode.
0	1	1	NPE-B Ethernet 0 is enabled in SMII mode. NPE-B Ethernet 1-3 is disabled and clock gated.

Table 234. NPE-C Ethernet Configuration Options

NPE-C Ethernet Fuse	NPE-C SMII Config Bit	Operation
0	0	NPE-C Ethernet is enabled in MII mode.
0	1	NPE-C Ethernet is enabled in SMII mode.
1	X	NPE-C Ethernet is disabled.



12.5.12 EXP_SMIIDLL

Register Name:		EXP_SMIIDLL													
Physical Address:		0xC400002C						Reset Hex Value:		0x00000000					
Register Description:		DLL bits for SMII used by the SMII DLL.													
Access: See below.															
3														1	
1														0	9
(Reserved)												exp_smiidll			

Register		EXP_SMIIDLL			
Bits	Name	Description	Reset Value	Access	
31:10	(Reserved)	(Reserved)	0x0	RO	
9:0	exp_smiidll	This field must be programmed to 0x18E before enabling SMII mode.	0x0	RW	

12.5.13 EXP_MST_CONTROL

Register Name:		EXP_MST_CONTROL																					
Physical Address:		0xC4000100				Reset Hex Value:				0x00000000													
Register Description:		Specifies values for bus arbitration priority, bus master locking, and external master parity support																					
Access: See below.																							
31																		5	4	3	2	1	0
(Reserved)																			ExtCfg	OddPar	Inpar_en	ArbMask #	GrantRemove

Register		EXP_MST_CONTROL			
Bits	Name	Description	Reset Value	Access	
31:5	(Reserved)	Reserved	0x0	RO	
4	ExtCfg	Allows External Masters to write EXP_INBOUND_ADDR register 0 = External master writes to EXP_INBOUND_ADDR are enabled 1 = External master writes to EXP_INBOUND_ADDR are ignored	0	RW	
3	OddPar	Odd or even parity is generated/compared on EX_PARITY for Inbound and Outbound accesses. 0 = Even parity on EX_PARITY 1 = Odd parity on EX_PARITY	0	RW	

12.5.14 EXP_INBOUND_ADDR

August 2006
Order Number: 306262-004US

Intel® IXP45X and Intel® IXP46X Product Line of Network Processors
Developer's Manual
717



Register		EXP_INBOUND_ADDR		
Bits	Name	Description	Reset Value	Access
31:21	BaseAddr	The register specifies the upper AHB address used for inbound transfers. Bits 31:24 of the AHB address reflect this register setting. Bits 23:21 of the AHB address are dependent on the value of AddrWidth. See description under AddrWidth.	0x0	RW
20:4	Reserved	Reserved	0x0	RO
3:0	AddrWidth	The value represents which bits of the BaseAddr should be used for the calculation of the AHB address on inbound transfers. 0x8: AHB ADDR = BaseAddr[31:24] & EX_ADDR[23:0] when EX_ADDR[24] = '0' 0x9: AHB ADDR = BaseAddr[31:23] & EX_ADDR[22:0] when EX_ADDR[23] = '0' 0xA: AHB ADDR = BaseAddr[31:22] & EX_ADDR[21:0] when EX_ADDR[22] = '0' 0xB: AHB ADDR = BaseAddr[31:21] & EX_ADDR[20:0] when EX_ADDR[21] = '0' others: Reserved	0x8	RW

17

Note: The BaseAddr must not be programmed to loop back to the Expansion Bus controller address space or Expansion Bus controller MMR space and results in unpredictable operation if programmed for loopback. Additionally, the Expansion bus controller does not support inbound accesses to the AHB Queue Manager SRAM. Accesses to the AHB Queue Manager queue's are allowed, however they must be 1-word accesses. Another restriction that must be followed is that if the BaseAddr is programmed to target the PCI Controller, the PCI controller cannot be programmed to access the Expansion bus at the same time. If simultaneous PCI to Expansion bus traffic and Expansion bus to PCI traffic is desired, there are two methods to support this.

1. Program the PCI Controller to transfer the data to main memory and then use the Intel XScale processor to transfer the data from main memory to the Expansion bus.
2. If there is PCI to Expansion bus traffic, ensure that there is never Expansion bus to PCI traffic until the PCI to Expansion bus traffic is complete.

The EXP_INBOUND_ADDR register can be read/written from the AHB bus and from an external master. This allows an external master to re-configure the BaseAddr if ExtCfg is clear. External masters must be careful not to program AddrWidth to an unintended value. External masters accesses to the EXP_INBOUND_ADDR register is only supported for word accesses. Sub-word accesses will be treated as a word access. Additionally, the EXP_BYTE_SWAP_EN is ignored for accesses to the EXP_INBOUND_ADDR register. Simultaneous writing of the EXP_INBOUND_ADDR register from the AHB and an external master results in unpredictable operation. If ExtCfg is set, external master reads to EXP_INBOUND_ADDR are supported, but writes are ignored. The Expansion bus controller uses the appropriate EX_ADDR bit to determine if the Expansion bus access targets an Expansion bus MMR or the AHB bus. If the most significant EX_ADDR bit (which is based upon AddrWidth) is a '1', the access will target the externally visible Expansion bus MMRs.

**Table 235. EX_ADDR Value to Access EXP_INBOUND_ADDR Register**

AddrWidth	EX_ADDR[3:0] to read / write EXP_INBOUND_ADDR	EX_ADDR[19:4] to read / write EXP_INBOUND_ADDR	EX_ADDR[24:20] to read / write EXP_INBOUND_ADDR
0x8	"01XX"	0x0000	"10000"
0x9	"01XX"	0x0000	"X1000"
0xA	"01XX"	0x0000	"XX100"
0xB	"01XX"	0x0000	"XXX10"

12.5.15 EXP_LOCK0

Register Name:		EXP_LOCK0	
Physical Address:	0xC4000108 and from External Master (See note below)	Reset Hex Value:	0x00000000
Register Description:	This register is intended to facilitate resource locking for multiple masters		
Access: See below.			
3 1			1 0
LOCK			

Register		EXP_LOCK0		
Bits	Name	Description	Reset Value	Access
31:0	LOCK	If the value of LOCK is 0x0 when read, the master owns the resource and the Expansion bus controller will set LOCK to 0xFFFFFFFF in the following cycle. The master owns the resource until the master writes a 1 to any of the bits in this register to clear the lock to 0x0. If LOCK is 0xFFFFFFFF when read, the resource is locked and the master does not own the resource and must re-read this register again to lock the resource.	0x0	RS/W1C

The intent of the EXP_LOCK0 register is to provide multiple masters a mechanism to lock some arbitrary hardware resource such as a particular region in DDR address space. When the EXP_LOCK0 register is read for the first time, the Expansion bus controller will return 0x0 to that master. If the value read from the EXP_LOCK0 register is 0x0, that master will have access to the resource that was locked until that master clears the lock by writing a '1' to any of the bits in the EXP_LOCK0 register. If another master reads the EXP_LOCK0 register before the original master clears the lock, the 2nd master will observe 0xFFFFFFFF, which means that the resource is locked and it does not have access to the resource. The IXP45X/IXP46X network processors do not have any hardware to prevent the 2nd master from accessing the locked resource and it is up to software to ensure that the 2nd master complies to this protocol. The 2nd master will continue to poll EXP_LOCK0 until it owns the lock (observes 0x0). Once the 2nd master is completed with the lock, it clears the lock to allow another master to obtain the locked resource. Only the master that owns the lock should be writing the EXP_LOCK0 register to clear the lock. Other masters must not write the EXP_LOCK0 register when it does own the lock.



The EXP_LOCK0 register can be read/written from the AHB bus and from an external master. External masters accesses to the EXP_LOCK0 register is only supported for word accesses. Sub-word accesses will be treated as a word access. Simultaneous reading of the EXP_LOCK0 register from the AHB and an external is allowed and the AHB master will obtain the resource lock.

Table 236. EX_ADDR Value to Access EXP_LOCK0 Register

AddrWidth	EX_ADDR[3:0] to read / write EXP_INBOUND_ADDR	EX_ADDR[19:4] to read / write EXP_INBOUND_ADDR	EX_ADDR[24:20] to read / write EXP_INBOUND_ADDR
0x8	"10XX"	0x0000	"10000"
0x9	"10XX"	0x0000	"X1000"
0xA	"10XX"	0x0000	"XX100"
0xB	"10XX"	0x0000	"XXX10"

12.5.16 EXP_LOCK1

Register Name:		EXP_LOCK1	
Physical Address:	0xC400010C and from External Master (See note below)	Reset Hex Value:	0x00000000
Register Description:	This register is intended to facilitate resource locking for multiple masters		
Access: See below.			
31			10
LOCK			

Register		EXP_LOCK1		
Bits	Name	Description	Reset Value	Access
0	LOCK	If the value of LOCK is 0x0 when read, the master owns the resource and the Expansion bus controller will set LOCK to 0xFFFFFFFF in the following cycle. The master owns the resource until the master writes a 1 to any of the bits in this register to clear the lock to 0x0. If LOCK is 0xFFFFFFFF when read, the resource is locked and the master does not own the resource and must re-read this register again to lock the resource.	0x0	RS/W1C

The EXP_LOCK1 register is identical in functionality to EXP_LOCK0, but can be used by software to lock a different arbitrary resource.

**Table 237. EX_ADDR Value to Access EXP_LOCK1 Register**

AddrWidth	EX_ADDR[3:0] to read / write EXP_INBOUND_ADDR	EX_ADDR[19:4] to read / write EXP_INBOUND_ADDR	EX_ADDR[24:20] to read / write EXP_INBOUND_ADDR
0x8	"11XX"	0x0000	"10000"
0x9	"11XX"	0x0000	"X1000"
0xA	"11XX"	0x0000	"XX100"
0xB	"11XX"	0x0000	"XXX10"

12.5.17 EXP_PARITY_STATUS

Register Name:		EXP_PARITY_STATUS																													
Physical Address:		0xC4000120										Reset Hex Value:								0x00000000											
Register Description:		Specifies the parity error status.																													
Access: See below.																															
31																													2	1	0
ErrorAddr																												InErrorSts	OutErrorSts		

Register		EXP_PARITY_STATUS			
Bits	Name	Description	Reset Value	Access	
31:2	ErrorAddr	AHB address[31:2] of the parity error. If multiple parity errors occur, ErrorAddr contains the address of the first parity error. After receiving a parity error, the ErrorAddr is locked until InErrorSts and OutErrorSts are both cleared by software.	0x0	RO	
1	InErrorSts	A parity error occurred on an inbound write. This bit will never get set if the PAR_EN bit is not set on EXP_MST_CONTROL register.	0	RW1C	
0	OutErrorSts	A parity error occurred on an outbound read. This bit will not get set unless PAR_EN is set in the EXP_TIMING_CS for which the parity error has occurred.	0	RW1C	

The OR of InErrorSts and OutErrorSts is generated to form exp_parity_error which is routed to the Interrupt Controller which can generate interrupts to Intel XScale processor. In the event of multiple inbound parity errors or multiple outbound parity errors, there is a race condition between when software performs a write to clear the EXP_PARITY_STATUS register and setting the InErrorSts/OutErrorSts error bit in hardware. If the software clears the error bit before another parity error of the same type (i.e Inbound), the EXP_PARITY_STATUS register will be set again. However if software clears the error bit on or after another parity error, the EXP_PARITY_STATUS register will be cleared. If the race condition described above is of concern, the best way to work around this issue is to ensure no other masters are targeting the Expansion bus controller while software is clearing the EXP_PARITY_STATUS register. The only way to stop outbound transactions is to have a software mechanism to stop the originating master from issuing transfers. The easiest mechanism to stop inbound

transfers is to clear ArbMask# bit in the EXP_MST_CONTROL register and then allow the current inbound transaction complete. To ensure the inbound transaction is complete, the Intel XScale processor can perform an outbound read to any Intel XScale processor general-purpose register (i.e ldr R4,[R2]) followed by a dependency on this register (i.e mov R4,R4).

12.5.18 EXP_SYNCINTEL_COUNT

[illegible]

Register		EXP_SYNCINTEL_COUNT		
Bits	Name	Description	Reset Value	Access
31:4	Reserved	Reserved	0x0	RO
3:0	Count	The count bits tell the Expansion bus controller how many clock cycles must elapse before the first data word is sampled from a Synchronous Intel StrataFlash Memory. The value that needs to be programmed is dependent on the Expansion bus clock frequency and needs to be set to the same value as the Latency Count bits defined in the Read Configuration Register in the Synchronous Intel StrataFlash Memory. Refer to the Synchronous Intel StrataFlash Memory datasheet for more information. 0000: Reserved 0001: Reserved 0010: Code 2 0011: Code 3 0100: Code 4 0101: Code 5 0110: Code 6 0111: Code 7 1000: Code 8 1001: Code 9 1010: Code 10 1011 - 1111: Reserved	0x0	RW



13.0 HSS Coprocessor

13.1 Overview

This document outlines the functional description of the HSS coprocessor.

The functionality supported by the High-Speed Serial (HSS) interfaces are tightly coupled with the code written on the Network Processor Engine (NPE) core. This chapter details the full hardware capabilities of the HSS interfaces contained within the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described below.

The HSS coprocessor enables the IXP45X/IXP46X network processors to communicate in a Time Divisible Multiplexed (TDM) bit serial fashion with external chips. The HSS interfaces are six-wire, serial interfaces that can operate at speeds from 512 KHz to 8.192 MHz.

The NPEs core controls each HSS interface. By programming certain parameters to the HSS interfaces — such as frame length/offset, frame signal polarity, and data endianness — the interfaces can be configured to support a variety of bit serial protocols. The bit streams protocols that the hardware supports are T1, E1, GCI, MVIP. The HSS hardware also has the ability to interface with certain xDSL framers. The HSS is the interface between the NPE Core and an external device (usually classified as a framer), which uses one of the above protocols.

For a list of supported protocols supported by the current software release, see the *Intel® IXP400 Software Programmer's Guide*.

Note: In this document, the NPE (Network Processing Engine) microprocessor core is also referred to as PSM or PSM2. The preferred term is NPE core.

Each HSS core contains 5 RX FIFOs, 4 RX FIFOs are for HDLC and are each two words in length, these are further divided into two buffers, which are each 1 word in length. These two buffers act in a ping-pong fashion.

The 4 FIFOs TX/RX can be configured to behave like two or one TX/RX FIFO, see [“FIFOs and Lookup Tables” on page 727](#) for more details. The TX FIFOs are identical in size and numbers to the RX FIFOs.

The fifth RX FIFO is for VOICE and is four words in length, this is also split into two buffers, which are each 2 words in length. These buffers also behave in a ping-pong fashion, so *the software processes two words (one buffer) at a time*.

Each HSS core can be programmed by the NPE Core to perform the byte/frame interleaving in the TX direction (MVIP), each core can also split up an incoming RX MVIP stream into its constituent voice/HDLC streams (using the lookup tables described below).

Two lookup tables 128*2 bits in size are present in each HSS core, 1 for the TX direction and the other for the RX direction. These tables indicate if an incoming/outgoing stream consists of HDLC/VOICE/56K or unassigned timeslots (bytes), the



length of the lookup table can be programmed as needed. Selecting a timeslot as HDLC tells the HSS core that the timeslot should be stored in a HDLC FIFO (the particular HDLC FIFO depends on what location in the lookup table was programmed, see [Figure 165 on page 728](#) for more details).

In the case where dual MVIP or normal E1/T1 protocols are operating, not all of the lookup table space will be used. The frame size programmed indicates the number of locations used in lookup table.

In the case of GCI, the lookup tables should be programmed to VOICE.

To support software ATM-TC (Transmission Convergence) operations, a HEC generator and an ATM scrambler are located within the HSS, note that only two are employed in total, 1TX and 1RX (not one per core).

13.1.1 High-Speed Serial Interface Receive Operation

Each High-Speed Serial Interface contains five receive FIFOs and four receive FIFOs intended for facilitating the use of HDLC. Each is two 32-bit words in length. The four receive FIFOs are further divided into two buffers, each one 32-bit word in length. These FIFOs are hardware support added to facilitate the implementation of four HDLC channels and do not preclude the implementation of more HDLC channels using the Network Processor Engine core and the HDLC coprocessor connected.

The HSS interface will be filling one buffer while the NPE Core empties the other buffer. The fifth receive FIFO is intended for voice-processing support and is four 32-bit words in length. This receive FIFO is split into two buffers, each buffer two 32-bit words in length.

These buffers also behave in a ping-pong fashion, so the NPE Core will read two 32-bit words at a time for processing. The location that each received byte is placed into these FIFOs is a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table will characterize each received byte as one of four types:

- Unassigned
- HDLC
- Voice
- 56-K mode

This characterization will be assigned on a time-slot basis using Intel-supplied APIs. For example, time slot 0 may be defined as a voice cell, time slot 1 as an HDLC wrapped packet, time slot 2 as an undefined time slot, and time slot 3 defined as an 56-K mode cell.

When the HSS receive interface processes the first byte (time slot 0), the look-up table will indicate that this received byte is a voice cell and needs to be placed into the Voice FIFO. Likewise, when the high-speed serial receive interface processes the second byte (time slot 1), the look-up table will indicate that this received byte is an HDLC cell and needs to be placed into one of the HDLC FIFOs. The actual FIFO the byte is placed in is dependent on the protocol implemented and the FIFO arrangement.

For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the high-speed serial receive interface processes the third byte (time slot 2), the look-up table will indicate that this received byte is an unassigned cell and needs to be discarded. When the high-speed serial receive interface processes the fourth byte (time slot 3), the look-up table will indicate that this received byte is a 56-K mode cell and will also be placed into the Voice FIFO.



13.1.2 High-Speed Serial Interface Transmit Operation

For transmission using the High-Speed Serial Interface, each High-Speed Serial Interface contains five transmit FIFOs, organized exactly as the receive FIFOs. (For additional details on the FIFO organization, see “[High-Speed Serial Interface Receive Operation](#)” on page 724.)

Each transmitted byte is placed into these FIFOs. The data is transmitted using the High-Speed Serial Interface as a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table will characterize each byte to be transmitted as one of four types:

- Unassigned
- HDLC
- Voice
- 56-K mode

This characterization will be assigned on a time-slot basis using Intel-supplied APIs.

Assume the same example as before, time slot 0 is be defined as a voice cell, time slot 1 is be defined as an HDLC wrapped packet, time slot 2 is be defined as an undefined time slot, and time slot 3 is be defined as an 56-K mode cell.

When the HSS transmit interface is ready to process the first byte (time slot 0), the look-up table will indicate that the byte to be transmitted is a voice cell and needs to be extracted from the Voice FIFO and placed onto the HSS interface. Likewise, when the HSS transmit interface is ready to process the second byte (time slot 1), the look-up table will indicate that the byte to be transmitted is an HDLC cell and needs to be extracted from one of the HDLC FIFOs and placed onto the HSS interface.

The actual FIFO the byte is extracted from is dependent upon the protocol implemented and the FIFO arrangement. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the HSS transmit interface processes the third byte (time slot 2), the look-up table will indicate that the byte to be transmitted is an unassigned cell. Using Intel-supplied APIs, the Intel XScale processor can program the HSS interface to transmit one of three values in an unassigned time slot:

- All zeros
- All ones
- High-impedance

When the HSS transmit interface processes the fourth byte (time slot 3), the look-up table will indicate that the byte to be transmitted is a 56-K mode cell and is located in the Voice FIFO. When the transmit interface detects from the transmit look-up table that the slot to be transmitted is a 56-K mode byte, only seven of the eight bits in a time slot will be valid. The most-significant bit or the least-significant bit will be invalid.

Using Intel-supplied APIs, the Intel XScale processor can program the invalid bit location as well as the value to be placed into the invalid bits location when data is transmitted. The data inserted into the invalid bit location can be programmed to be logic 0, logic 1, or tri-state. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

13.2 Feature List

- T1/E1/GCI/MVIP protocols supported in hardware.
- HSS coprocessor split into ‘cores’, each core houses a set of FIFOs, stream generator and stream parser. This accommodates re-use of HSS design.



- Ten FIFOs per HSS core, 5 for TX, 5 for RX (1 voice, 4 HDLC).
- The four HDLC FIFOs can be programmed to operate as either 2 FIFOs or 1 FIFO.
- Two lookup tables are employed per HSS core (1 for TX, the other for RX), this allows voice/HDLC channelization, also removes unassigned timeslots.
- Condition signals are used to indicate the status of the FIFOs with the HSS.
- An ATM-TC scrambler and HEC generator are supplied within the HSS.
- The line clock speed is programmable (allowing differing protocols speeds).
- Loopback facility available for debug purposes.
- Very flexible frame generation options (allows HSS to interface to many framers), the following are the programmable options,
 - The frame pulse can be set active low, active high, falling edge or rising edge.
 - The frame pulse and sync clock can be set as an input or an output.
 - The sync clock can be set to positive or negative edge with respect to (w.r.t.) the data.
 - The sync clock can be set to positive edge or negative edge with respect to the frame pulse.
 - The data polarity can be set low or high.
 - The data endianness can be programmed (w.r.t. what's transmitted or received first).
 - Open drain mode on the tx data pins.
 - The clock can run at the data rate or double the data rate.
 - The output data can be high impedance, high or low when not driving data.
 - The NPE Core can select to use FBit or not.
 - The frame size can be programmed; the maximum value is 1,024 bits.
 - The frame pulse offset can also be programmed; the maximum value is 1,023 bits.
 - The NPE Core can detect if an unexpected frame pulse has been received (in both TX and RX) by the HSS.
 - The clock speed can run at 512 KHz, 1.536 MHz, 1.544 MHz, 1.568 MHz, 2.048 MHz, 4.096 MHz, 8.192 MHz depending on the protocol which is in use.

13.3 Theory of Operation

The external device communicates with the HSS coprocessor using three signals per direction, namely a frame pulse, a clock, and data bit.

The data stream consists of frames, the number of frames depends on the protocol in use. Each frame is composed of timeslots, each of these timeslots consists of 8 bits, the number of the timeslot indicates its location within the frame, this timeslot count value is maintained within each HSS core and the NPE Core has access to this information. Some protocols use an FBit (frame bit) at the beginning of the frame (T1). In the case where T1 is running at 1.544 MHz, the HSS will read in the received FBit, add on 7 padding bits (zeros in the 7 lsbs) and place the byte into the voice FIFO. The software will read the FIFO location like any other timeslot. When MVIP is used where T1s are located within the MVIP frame (see [“MVIP” on page 742](#)), the HSS will not process the FBits any differently to other timeslots, and will treat them as dictated by the lookup tables (assign them as either voice/HDLC or unassigned).



The number of timeslots expected by the HSS core is programmable by the NPE Core. The maximum frame size is 1024 bits, the maximum frame pulse offset is 1,023 bits.

To support ATM-TC, a LFSR scrambler is also located in the HSS. The NPE Core writes an initial value to the LFSR register. The NPE Core then reads back the register. The MSb (bit 31 of the bus to the NPE Core) is an XOR operation of bits 30 and 27 of the LFSR register. The NPE Core also indicates when the LFSR register should be updated.

A HEC generator is also included. The NPE Core is used to initialize the HEC register.

13.3.1 FIFOs and Lookup Tables

13.3.1.1 FIFOs

For every HSS core instantiated within the HSS coprocessor, there are 5 RX FIFOs and 5 TX FIFOs as described in [Section 13.1, “Overview” on page 723](#).

In the TX direction, a buffer will be emptying into the external device while the other buffer will be filled by the NPE Core. The HSS will keep count of the timeslot value w.r.t. the programmed frame length. When the software reads the TX timeslot value, that value represents the timeslot due to be written by the NPE Core into the TX buffer. The NPE Core does not need to know the locations in the buffer that contain the relevant data, as the HSS will automatically increment the buffer location. When the HSS detects that the TX buffer location has been filled, the `hss_tx_va_empty` / `hss_tx_vb_empty` / `hss_tx_h_empty` signal will go low indicating that the buffer is now full. The NPE Core should not attempt to write more data until the HSS indicates an empty buffer.

In the case of a full RX buffer, the software is expected to read to the end of the buffer. If more read instruction are issued after the end of the buffer is reached, the pointer will not go back to the start of the buffer. The pointer will go back to the beginning of the RX buffer when that RX buffer has been re-filled with received data.

When the HSS needs to process byte interleaved dual MVIP (2 E1s or 2 T1s are present in the stream), the 4 HDLC FIFOs should be programmed (by the NPE Core) to resemble 2 FIFOs, each of size 4 words (2 words per buffer). The software services two words when the `hdlc full/empty hdlc` flag asserts.

If E1/T1/GCI operation is desired, then the four HDLC FIFOs can be programmed to act as one FIFO of size eight words (software services four words). In frame interleaved MVIP, the HDLC FIFOs should be programmed to resemble one FIFO also.

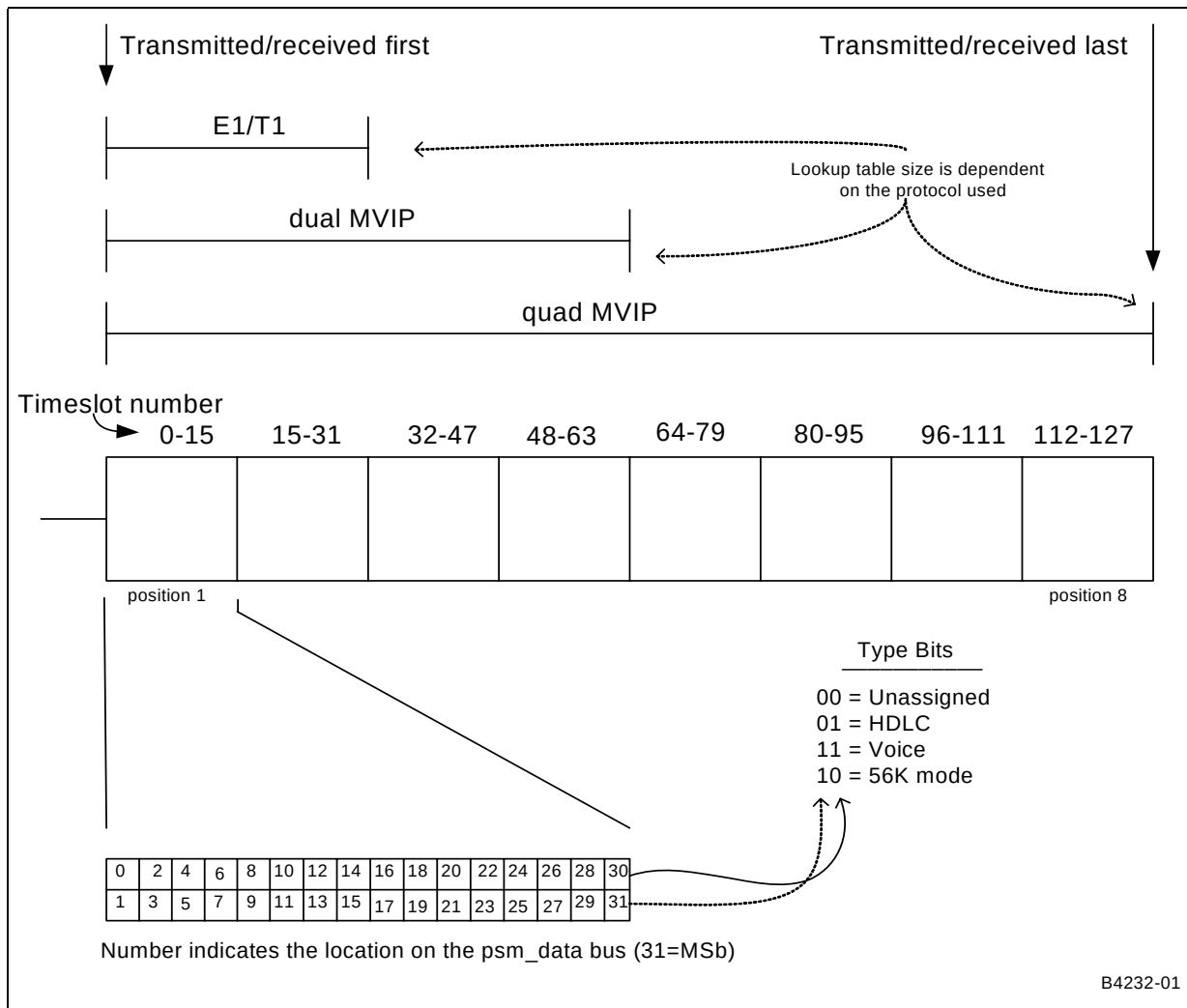
In byte interleaved quad MVIP, the `hdlc` buffers must be programmed to act like four FIFO. Each FIFO contains two words, the software services one word only.

The voice FIFO contains four words, two words per buffer. The software services two words when the appropriate voice condition flag asserts.

13.3.1.2 Lookup Tables

Two lookup tables (each 2*128 bits) are present in every HSS core (one for TX, the other for RX). The tables tell the HSS core what timeslots are HDLC/voice or unassigned within a stream.

Each lookup table requires eight write instructions (32 bit) to be completely filled by the NPE Core. This allows each timeslot to be programmed by the NPE Core to be either unassigned/voice/HDLC/56Kmode. For more details on 56Kmode, see [“56K Mode” on page 733](#). [Figure 165](#) illustrates how each lookup table is organized.

Figure 165. Look-Up Table Organization


In [Figure 165](#), the first NPE Core write to the LUT will be to position 1, the last write will be to position 8.

Each FIFO controller will communicate to the arbitrator any full RX buffers and any empty TX buffers, when the TX FIFO (both buffers) is empty (under-run), and when the RX FIFO (2 buffers) is full (over-run). It is up to the NPE Core to prevent the TX FIFO from underflowing and prevent the RX FIFO from overflowing.

In the case where the HSS has finished transmitting the first TX buffer and the second buffer is due for transmission, the second buffer must have been completely filled previously by the NPE Core to prevent underflow. The same system applies to the RX FIFOs.

In the case of 1 or more simultaneous RX voice 'A' buffers becoming full, the arbitrator will generate the 'hss_rx_va_full' signal as needed. The NPE Core will issue the HSSrdRxVaCond instruction (result indicates which FIFO/core activated the signal) and



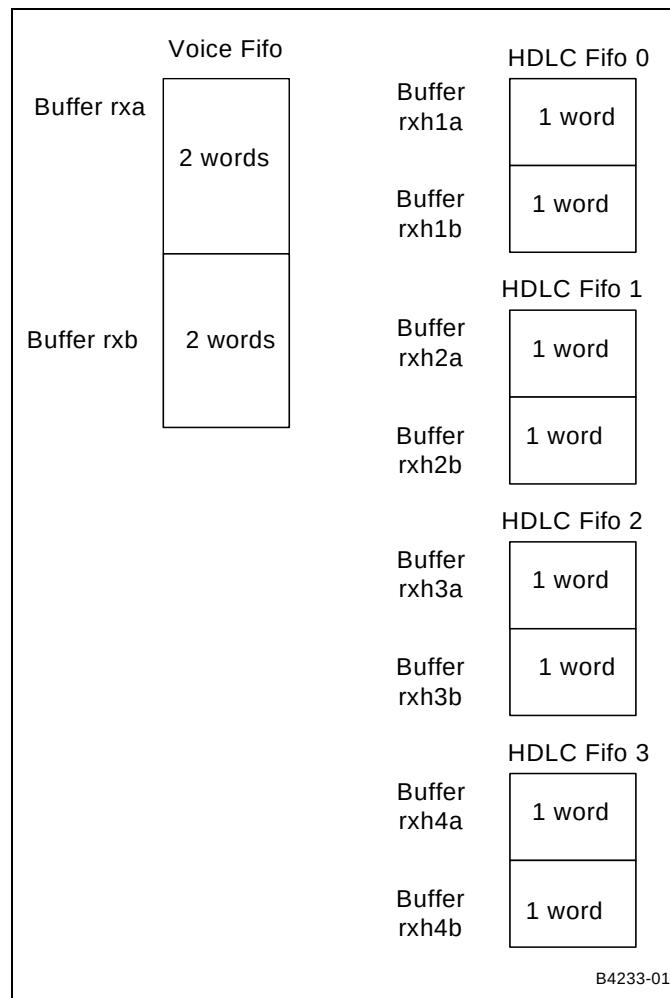
issue instructions as appropriate. As the HSS knows which core/buffer the NPE Core is going to service. The HSS will ensure that the FIFO related instructions will be directed towards the correct buffer.

It is necessary to have eight buffer pointers per direction in each HSS core. Their usage is described as follows:

- Four of these pointers allow external devices to write to 4 HDLC buffers (RX HDLC).
- One is for allowing an external device to write to the voice buffer (RX voice).
- Three are for allowing the NPE Core to read from a HDLC/voice buffer (RX).
- Four of these pointers allow external devices to read from 4 HDLC buffers (TX HDLC).
- One is for allowing an external device to read from the voice buffer (TX voice).
- Three are for allowing the NPE Core to write to a HDLC/voice buffer (TX).

None of these pointers are read/writable by the NPE Core.

Figure 166. HSS Core RX Buffer Structure (Identical to TX Buffer Structure)

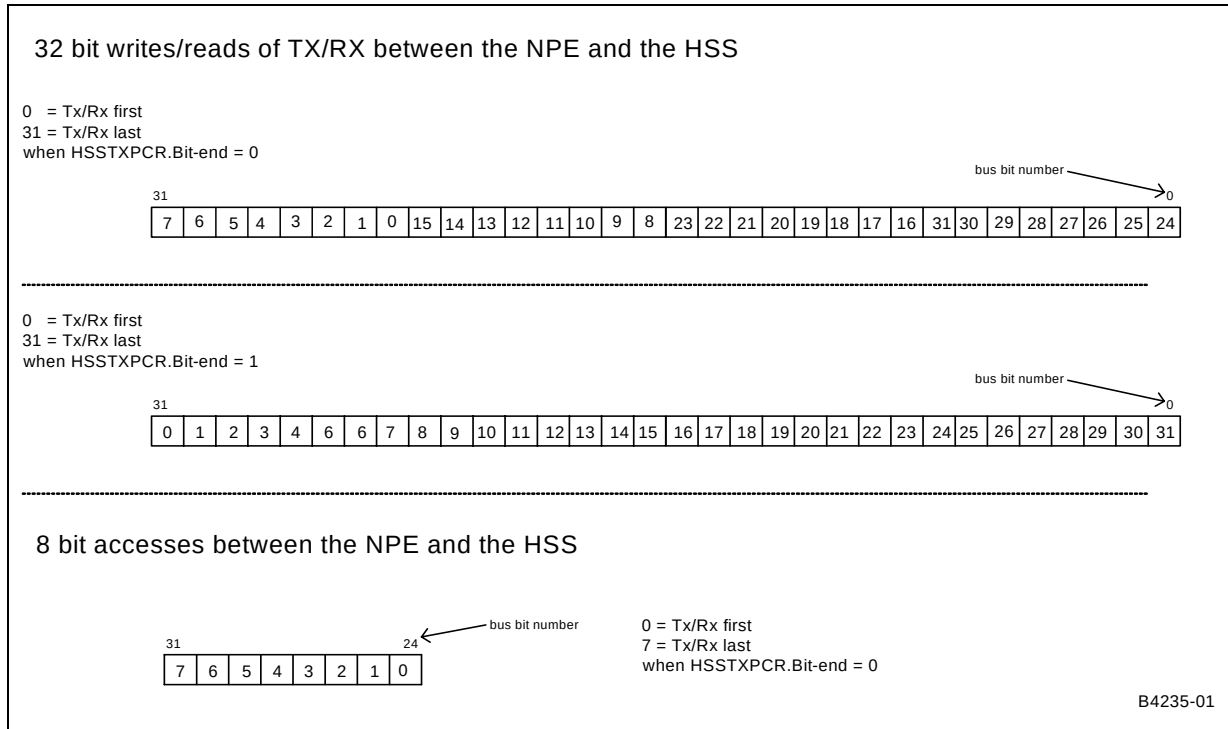




13.3.2 Endianness

The endianness of the data bus must be taken into account when writing data to the HSS for transmitting. The same goes for reading back received data.

Figure 167. HSS Endianness Examples



Changing the endianness only affects data to/from the FIFOs within the HSS, writes to the lookup tables are not affected.

When an 8-bit read is performed, the byte read will always be the byte received first. The same principle applies to TX data written to the HSS by the NPE Core.

13.3.3 Programmable Frame Pulse Offset and Frame Synchronization

An offset can be programmed by the NPE Core between the frame pulse and the start of the frame.

Before the HSS indicates to the NPE Core that it has received data, it must synchronizes to the frame pulse.

The HSS must detect two frame pulses (in the case of gapped frame pulses, then on the correct detection of the second frame pulse) to gain sync. RX frame pulses before must be synchronized before any data is placed into the Rx FIFOs.

If there is no Rx offset (programmed value of 0), then the data at the start of the second frame will be considered valid data and will be placed into the appropriate FIFOs (lut dependent). If there is an offset programmed, then the first data at the beginning of the third frame will be considered as the start of valid data. Either way, the first data the NPE Core receives will begin at the start of a frame. Therefore, no timeslot counters are needed within the HSS.



The HSS will not request data from the NPE Core until it has synced up to the TX frame pulse. Meaning that the HSS must detect two consecutive frame pulses (in the case of gapped frame pulses, then sync is assumed on the detection of the second frame pulse). Once data has been sent from the NPE Core, the data will be transmitted at the start of the next frame. If an offset is programmed, the data transmitted will be transmitted the offset number of clock cycles before the frame pulse.

If the HSS core has not synchronized to the frame pulse, it will not request TX/RX servicing nor will it indicate under/overflow conditions. Data received before synchronization is lost, regardless of what caused the loss of sync (reset/frame pulse error/incorrect programming, and so on), will be dropped. When not synchronized, the TX data pin is set to high impedance.

If under-run/over-run occurs, frame sync is maintained if the frame pulse is still generated (if sourced either internally or externally).

The behavior of the HSS is indifferent to the source of the frame pulse (in terms of synchronization), be it sourced externally or generated internally.

Figure 168. TX Frame Sync Example (Presuming Zero Offset)

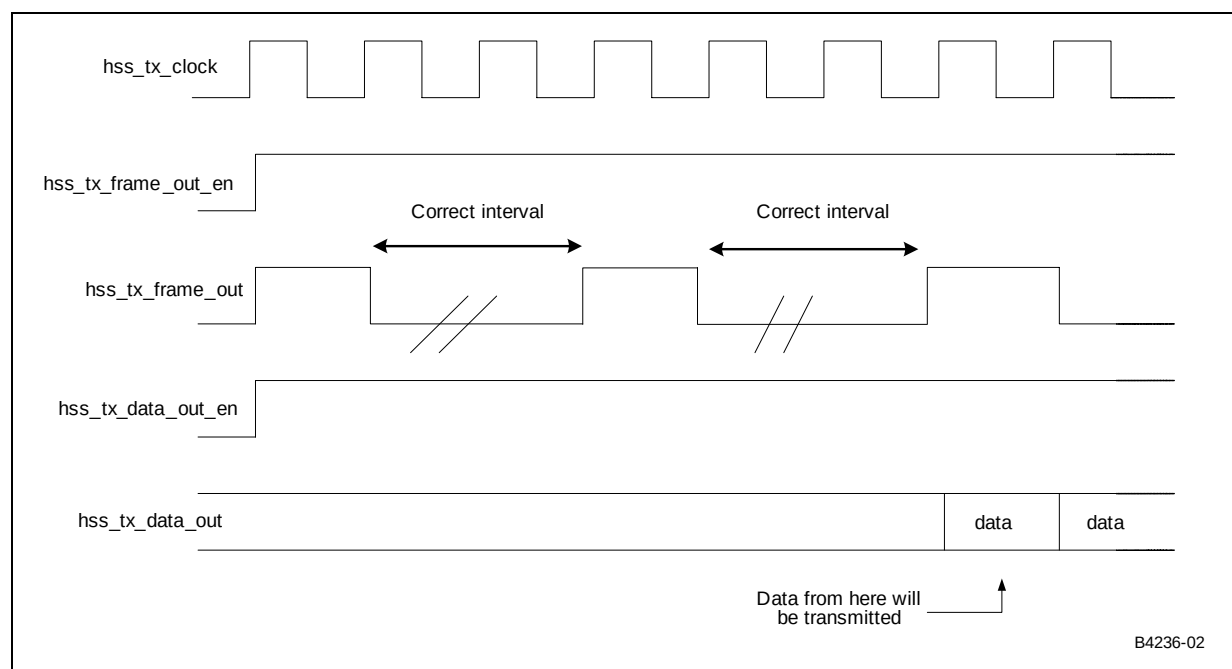
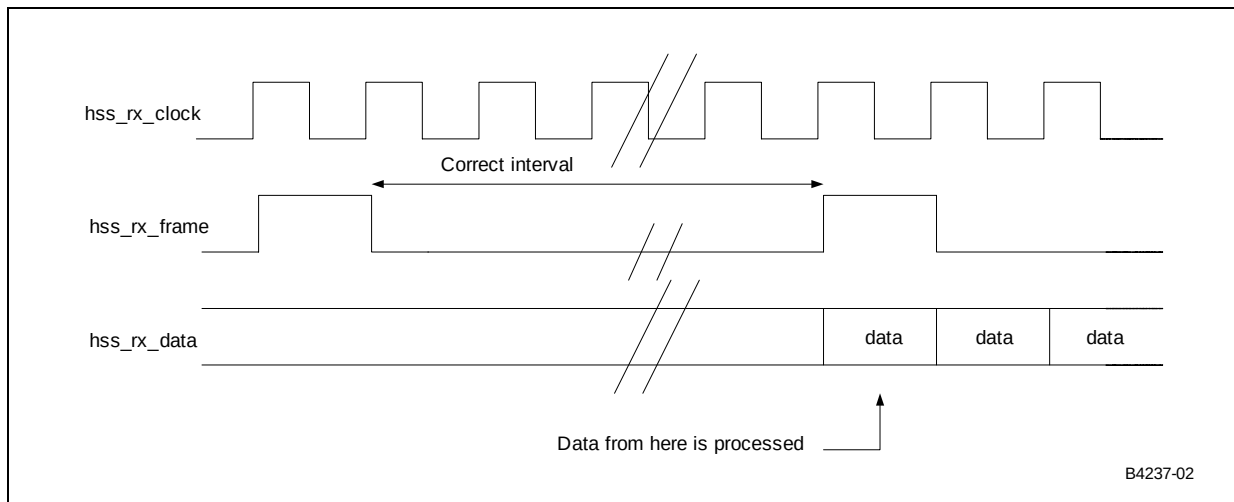


Figure 169. FRX Frame Sync Example (Presuming Zero Offset)


13.3.4 Underflow / Overflow / Unexpected Frame Pulse

Underflow occurs if the HSS attempts to read from a FIFO which has not been filled by the NPE Core.

If this occurs in a HSS core, then all transmission in that core in that direction is halted and an error condition is indicated to the NPE Core. Once the NPE Core has acknowledged that error condition (hss_error condition signal), the HSS core will wait until the beginning of the next frame to request for data. That data will be transmitted on the start of the following frame. In the case where the 2 HSS cores simultaneously underflow, HSS core 0 always gets priority.

Within each core can be multiple FIFOs, 5-2 FIFOs (4 hdlc+2voice to 1 hdlc + 1 voice). In the case where a number of these FIFOs underflow in around the same time, the first FIFO to underflow will be the one indicated to the NPE Core. When the NPE Core reads the error register, all FIFO errors and the FIFOs themselves in the HSS core for that direction are cleared. Upon reading the error register, the tx condition signals from the HSS core are cleared. The conditions signals will go high again when data is needed for transmission.

If synchronization is attained but the NPE Core withholds sending data to the HSS TX FIFOs for many frames, the HSS will not indicate that as an error and the TX condition flag will remain asserted.

Overflow occurs if the HSS attempts to write to a FIFO that has not been emptied by the NPE Core. If this occurs, then all reception in that HSS core is halted until the NPE Core acknowledges this condition by reading the error register. Once this has occurred, the HSS will start filling the RX FIFOs at the start of the next frame (offset compensated).

In the case where the two HSS cores overflow simultaneously then HSS core 0 will always be given priority. Once serviced, HSS core 1 is then allowed to indicate its overflow error to the NPE Core.

Within each core can be multiple FIFOs, 5-2 FIFOs (4 hdlc+2voice to 1 hdlc + 1 voice). In the case where a number of these FIFOs overflow in around the same time, the first FIFO to overflow will be the one indicated to the NPE Core. When the NPE Core reads



the error register, all FIFO errors and the FIFOs themselves in the HSS core for that direction are cleared. When the NPE Core reads the error register, the RX condition signals are cleared.

The HSS can indicate TX and RX errors in the same HSS core or different HSS core simultaneously. Meaning that the HSS can indicate an HSS core 0 RX and HSS core 1 TX error simultaneously. During under/overflow the tx data pin is set to the programmed unassigned value.

Unexpected frame pulse error is defined as a frame pulse that occurs where one should not exist (incorrect frame length, frame pulse offset incorrectly programmed, and so on). If this happens simultaneously with an over/underflow error, the unexpected frame pulse is given priority.

After the NPE Core reads the error register, the FIFOs in that direction and the condition flags to the NPE Core are cleared.

A frame pulse not present where one is expected is not considered as an error and normal operations are maintained.

Note: Error conditions are not signaled to the NPE Core while that core is not synchronized to the frame pulse.

13.3.5 56K Mode

Certain protocols require “bit stealing”, meaning certain bits are dropped (do not carry information). In E1 mode for example, the HSS transports at 2.048Kbps which gives 64K per channel (32 channels). 56K mode places a value (default = 0) in the MSb of each timeslot transmitted (depending on the endianness it can be at the left or right side of the byte).

As this value in the MSb does not carry information about any of the channels, it therefore reduces the capacity per channel to 56 Kbps,

(8000 frames * 32 timeslots per frame * 7 bits per timeslot / 32 channels = 56 Kbyte per channel).

The value placed by the HSS into the MSb is programmable by the NPE Core. The HSS performs this by overriding FIFO control for that bit in that timeslot, thus preventing FIFO data from been accessed.

The 56K mode has no effect on RX data, except that it is presumed that the transmitter at the far end of the line has performed 56K operations on the stream. Received “stolen” bits are placed in the RX FIFO and are treated as data by the HSS.

13.3.6 Frameless Data Protocol Support

The HSS doesn't need to have a frame pulse supplied in order to transmit/receive data. In this case, the HSS should have its lookup tables/frame lengths/programmed as if it was using the 512-KHz GCI protocol. The HSS immediately assumes synchronization and uses internal logic that predicts frame pulses (for gapped frame pulses) to coordinate transmission and reception of data. The internal frame generator is not used nor is it sent out on the frame pin (for either TX/RX).

13.3.7 Loopback

Loopback is a debug function that can be used to deduce and debug problems observed with the system.



It can help isolate a problem by eliminating the HSS coprocessor and all blocks further up the line as a source of a problem, if the system works correctly in loopback mode this indicates that a problem (if one exists) is within blocks further down the system.

All TX and RX FIFOs should be empty before loopback is attempted. The HSS must synchronize to the frame pulse before any operations commence. The user is free to select either an internal or external frame pulse and clock.

When using the internal frame pulse/clock, the TX frame pulse/clock is internally looped over to the RX side. If using the external frame pulse/clock, the clock/frame pulse must be supplied on the TX frame pulse/clock pins, this is then automatically sent to the RX logic also, there is no need to supply any signals on the rx pins.

No data is sent or received to/from the outside world while in loopback mode. This mode can only be directly activated by the NPE Core.

13.4 HSS Registers and Clock Configuration

There are numerous Control and Status Registers (CSRs) contained within the NPE/HSS interface of the IXP45X/IXP46X network processors that are used to configure the many unique HSS settings discussed in this manual. The functional details of these registers are not exposed within any documentation for the IXP45X/IXP46X network processors and are reserved for use by NPE firmware only. The `IxHssAcc` API, however, provides indirect access to these registers to allow the complete control and configuration of all HSS features enabled by a particular IXP400 software. The *Intel® IXP400 Software Programmer's Guide* should be referenced for specific information regarding use of the `IxHssAcc` API.

There is one register titled the HSS Clock Divider Register that provides a means to generate a unique data clock for each of the two HSS interfaces for the IXP45X/IXP46X network processors. The `IxHssAcc` API will configure the HSS clock divider register with the appropriate values depending on which clock frequencies is selected, being 512 KHz, 1.536 MHz, 1.544 MHz, 2.048 MHz, 4.096 MHz, and 8.192 MHz. This is discussed further in the next section:

13.4.1 HSS Clock and Jitter

Each of the two High-Speed Serial (HSS) interfaces on the IXP45X/IXP46X network processors can be configured to generate an output clock on the `HSS_TXCLK` and `HSS_RXCLK` pins. This output clock, however, is not as accurate and stable as using an external oscillator. That is because the HSS clock is based on the internal, 133.32-MHz AHB bus. This frequency does not divide down easily. Subsequently, jitter and error are introduced into the resultant HSS output clock.

If developers are clocking a framer, DAA, or other device with a sensitive input PLL, they should use an external clock.

To provide developers with additional data, this chapter contains five tables:

- [Table 238, "HSS Tx/Rx Clock Output" on page 735](#)
- [Table 239, "HSS Tx/Rx Clock Output Frequencies and PPM Error" on page 735](#)
- [.....Table 240, "HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization" on page 735](#)
- [Table 241, "HSS Frame Output Characterization" on page 736](#)
- [Table 242, "Jitter Definitions" on page 736](#)

The jitter and error calculations, in the following tables, are valid only for IXP400 software Release 1.4 and later.



13.4.2 Overview of HSS Clock Configuration

The HSS clock can be configured by using the HSS API in IXP400 software releases 1.4 and later. The clock speed can be set by using the HSS API.

It is important to be aware of the jitter and frequency error on the output clock of the HSS. The six clock values in [Table 238](#) have been pre-defined by IXP400 software release 1.4 to minimize jitter and PPM error. The six clock values can only be selected by using the HSS API.

Table 238. HSS Tx/Rx Clock Output

HSS Clock Output Frequency	Protocol	Frame Size (Bits)	Notes
512 KHz	GCI	32	1
1.536 MHz	GCI	96	1
1.544 MHz	1 T1	193	
2.048 MHz	1 T1/E1	256	2
4.096 MHz	2 T1/E1	512	2
8.192 MHz	4 T1/E1	1,024	2
Notes: 1. These clock speeds are supported using the HSS API. However, the GCI protocol is not supported by the Intel® IXP400 Software. 2. When the frequencies of 2.048, 4.096, or 8.192 MHz are used for T1, the data rate for each T1 remains at 1.544 MHz by making certain time slots within a frame unassigned and with no data. The HSS for the IXP45X/IXP46X network processors can be configured to discard unassigned time slots.			

Table 239. HSS Tx/Rx Clock Output Frequencies and PPM Error

HSS Tx/Rx Frequency	Min. Frequency (MHz)	Avg. Frequency (MHz)	Max. Frequency (MHz)	Avg. Frequency Error (PPM)	Notes
512 KHz	0.508855	0.512031	0.512769	-60.0096	1, 2, 3
1.536 MHz	1.515	1.536	1.55023	-60.0096	1, 2, 3
1.544 MHz	1.515	1.5439	1.55023	+60.0024	1, 2, 3
2.048 MHz	2.01998	2.0481	2.08313	-60.0096	1, 2, 3
4.096 MHz	3.92118	4.0962	4.16625	-60.0096	1, 2, 3
8.192 MHz	7.4066667	8.1925	8.3325	-60.0096	1, 2, 3
Notes: 1. These HSS Tx/Rx clock output frequencies are based on the set-up parameters in Table 238 . 2. Characterization data of the HSS Tx/Rx clock output frequency data was determined by silicon simulation. 3. The parts-per-million (PPM) error rate is calculated using the average output frequency versus the ideal frequency.					

Table 240. HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization

HSS Tx/Rx Clock Output Frequency	Pj Max. (ns)	Cj Max. (ns)	Aj Max. (ns)
512 KHz	12.189	15	18.283
1.536 MHz	9.063	15	86.102
1.544 MHz	12.359	15	210.099
Note: For jitter definitions, see Table 242 on page 736 .			

**Table 240. HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization**

HSS Tx/Rx Clock Output Frequency	Pj Max. (ns)	Cj Max. (ns)	Aj Max. (ns)
2.048 MHz	-8.204	15	118.957
4.096 MHz	10.9	15	190.742
8.192 MHz	12.951	15	226.634
Note: For jitter definitions, see Table 242 on page 736 .			

Table 241. HSS Frame Output Characterization

HSS Tx/Rx Frequency	Frame Size (Bits)	Actual Frame Length (μs)	Frame Length Error (PPM)
512 KHz	32	62.496249	-60.0096
1.536 MHz	96	62.496249	60.016
1.544 MHz	193	125.007499	60.0024
2.048 MHz	256	124.9925	-60.0096
4.096 MHz	512	62.496	-60.0096
8.192 MHz	1024	62.49624	-60.0096
Note: PPM frame length error is calculated from ideal frame frequency.			

Table 242. Jitter Definitions

Jitter Type	Jitter Definition
Period Jitter (Pj):	$Pj_{(i)} = Period_{(i)} - Period_{average}$
Cycle-to-Cycle Jitter (Cj):	$Cj_{(i)} = Pj_{(i+1)} - Pj_{(i)}$
Wander-or-Accumulated Jitter (Aj):	$Aj_{(i)} = \sum_i Pj$

13.5 HSS Supported Framing Protocols

The following sections provide an overview of some Framing Protocols supported by each of the HSS interfaces for the IXP45X/IXP46X network processors, in addition to the recommended HSS interface setting for each protocol that are configured using the IxHssAcc API defined in the *Intel® IXP400 Software Programmer's Guide*.

13.5.1 T1

T1 is a serial data stream operating at 1.544 MHz. The stream is composed of frames of which there are 8000 a second. Each frame consists of 1 frame bit and 24 time slots, each time slot is a byte in size. As there are 24 time slots per frame, T1 can carry 24 channels.

[Figure 170](#) and [Figure 171](#) illustrate a typical transmitted T1 frame with an active high frame sync (level) and a positive edge clock for generating data. The HSS clock and frame pulse can be programmed to be either HSS outputs or HSS inputs. [Figure 170](#) shows an example with an internally generated frame pulse. [Figure 171](#) shows an example with an externally generated frame pulse. An offset can be programmed indicating when the TX frame is to be transmitted. The Polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

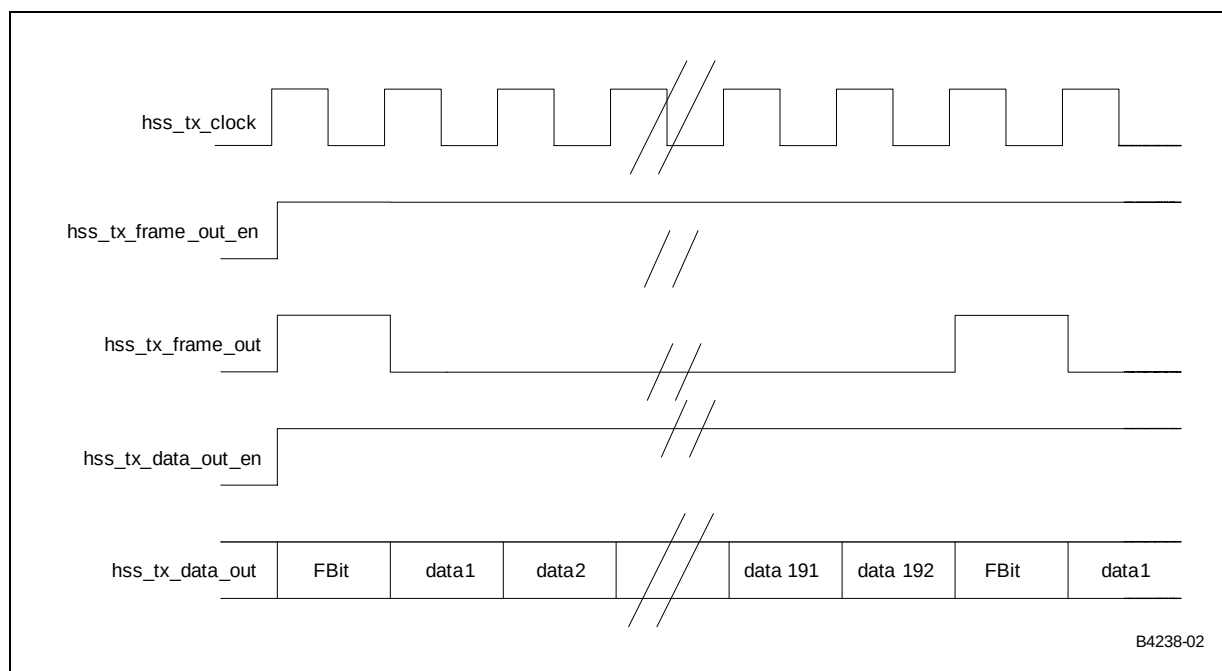
**Figure 170. T1 TX Frame, HSS Generating Frame Pulse**

Figure 170 illustrates a typical T1 frame with active high frame sync (level) and a posedge clock for generating data. If the frame pulse was generated with a negedge clock, the frame pulse in Figure 170 would be located one half clock space to the right. The same location applies to the data when being generated on the negedge of the clock.

In Figure 170 and Figure 171, the FBit to be transmitted is stored in the HSS Transmit FIFO. The HSS knows which time slot in the FIFO is holding the F Bit, as it knows from the time slot counter and frame offset when the F Bit should be transmitted.

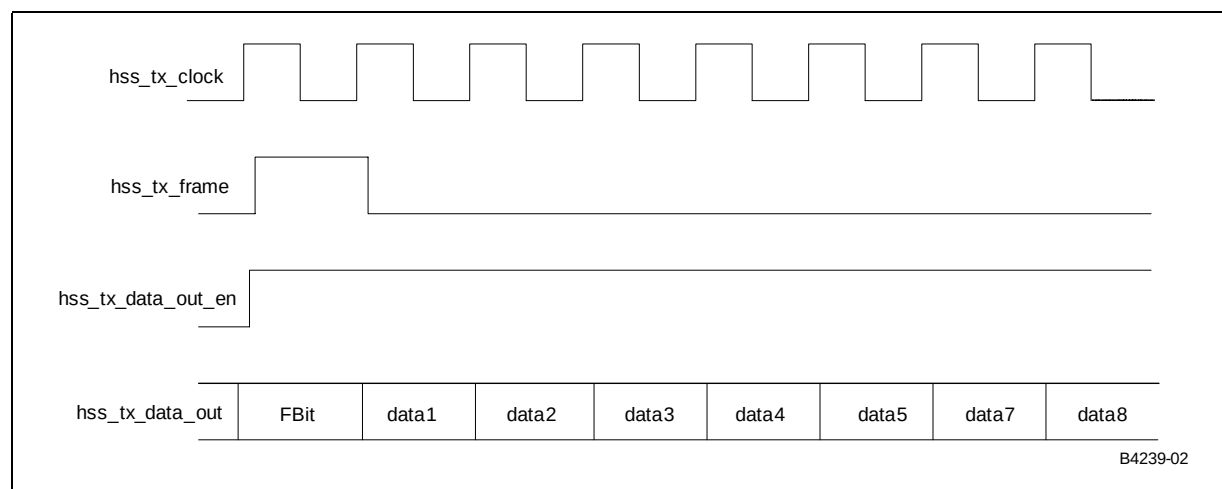
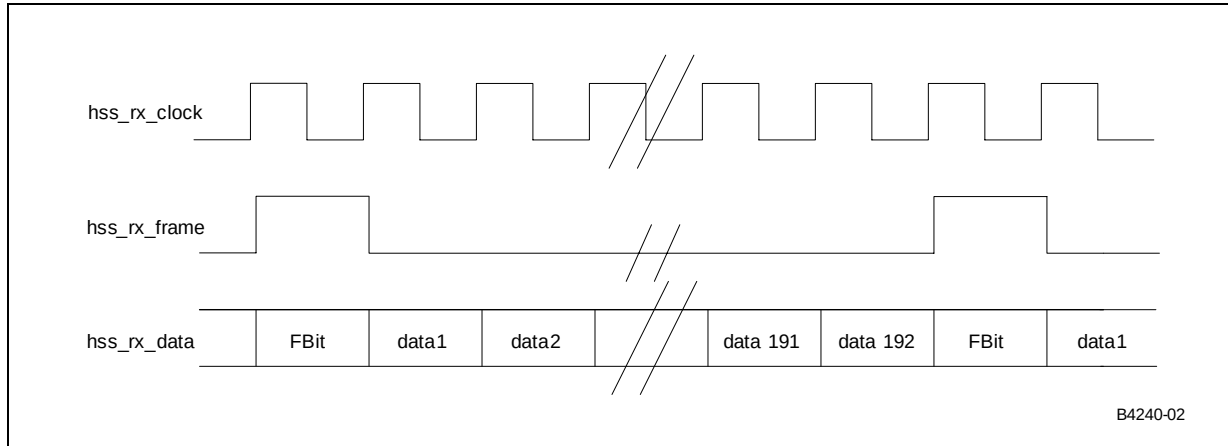
Figure 171. T1 TX Frame Using External Frame Pulse

Figure 172 illustrate a typical T1 received frame with an active high frame sync (level) and a positive edge clock for sampling data.

In [Figure 172](#), as stated in [Section 13.3, “Theory of Operation” on page 726](#), the FBit to be received is padded with 7 other bits (zeros) and placed into the HSS Receive FIFO.

Figure 172. T1 RX Frame Using External Frame Pulse



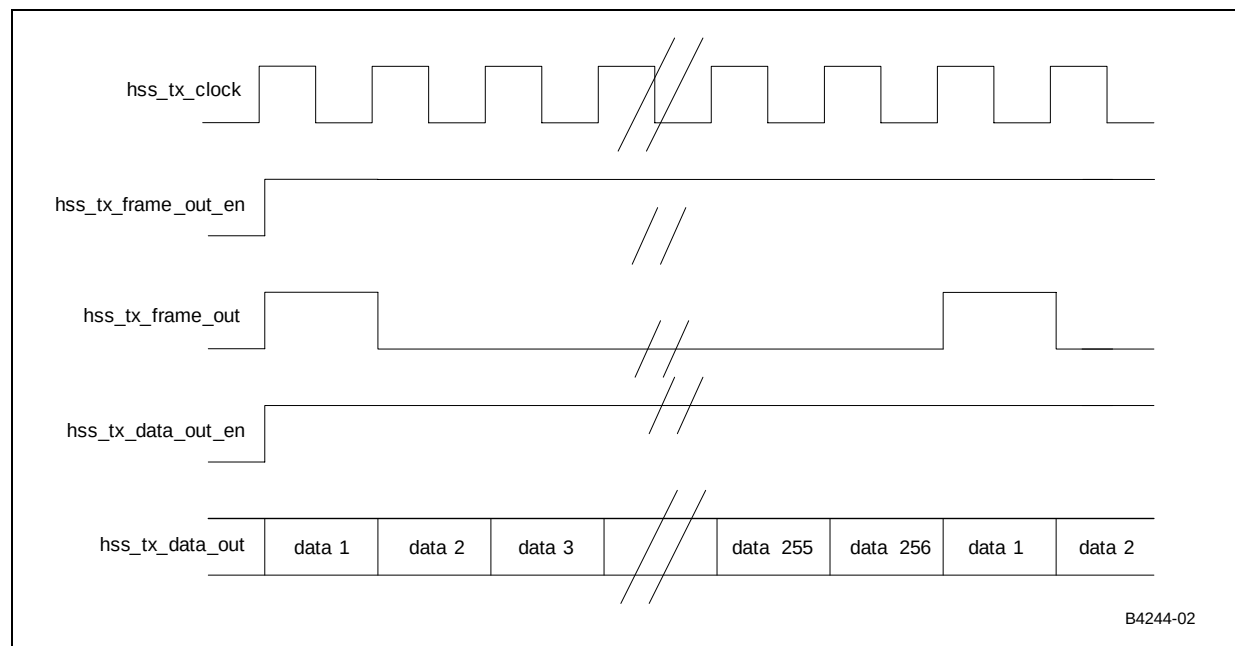
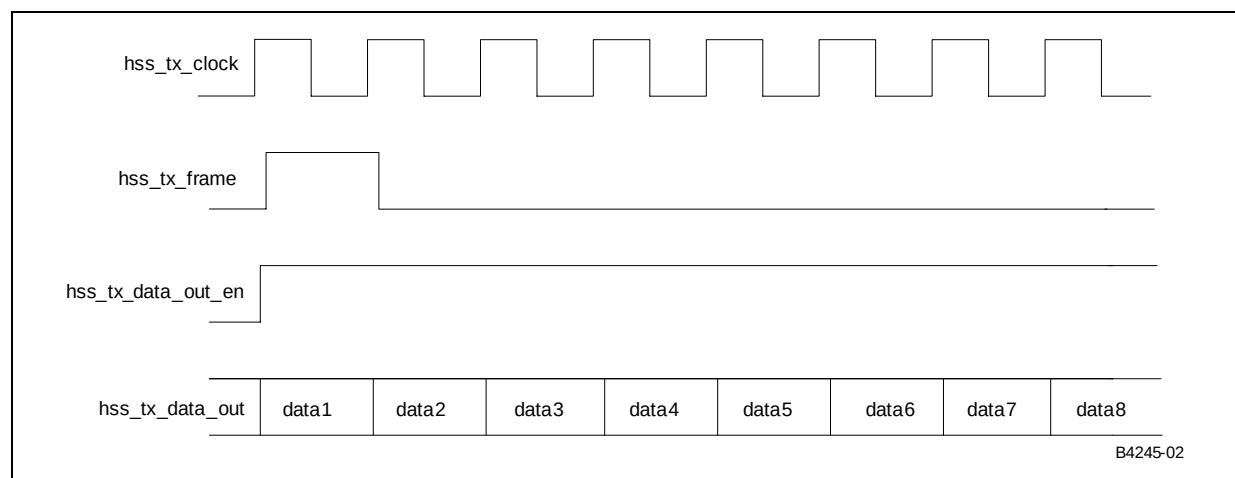
By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for T1 operation:

- Frame size 193 bits (for T1).
- Frame sync simultaneous with FBit – TX frame offset and RX frame offset should be set due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to use FBit (not data for T1) at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

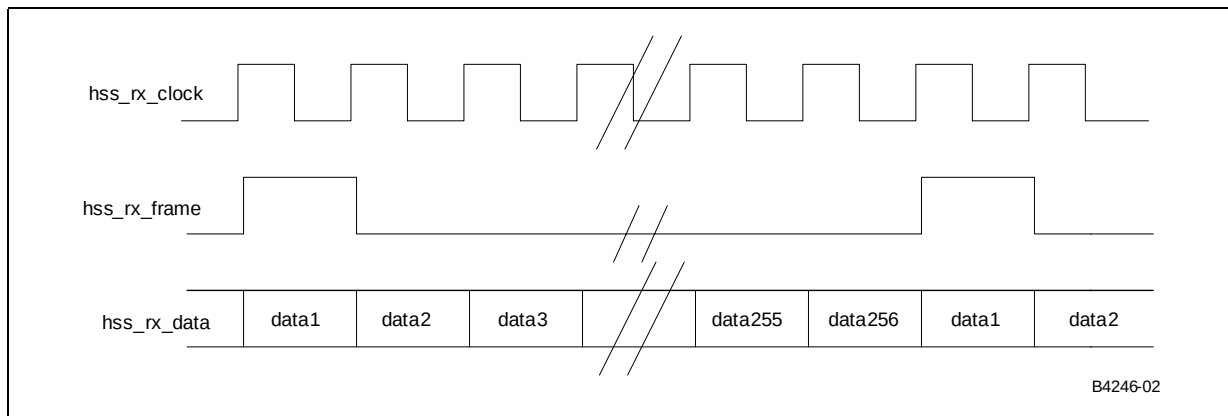
13.5.2 E1

This is a high-speed serial stream operating at 2.048 MHz. The stream is composed of frames of which there are 8000 a second. Each frame consists of 32 slots, each slot is a byte in size. As there are 32 slots per frame, E1 can carry 32 channels. There are no frame bits in this protocol.

[Figure 173](#), [Figure 174](#), and [Figure 175](#) illustrate a typical E1 frame with an active high frame sync (level) and a positive edge clock for generating or sampling data. The HSS clock and frame pulse can be programmed to be either HSS outputs or HSS inputs. An offset can be programmed indicating when the TX frame is to be transmitted. The polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

**Figure 173. E1 TX Frame, HSS Generating Frame Pulse****Figure 174. E1 TX Frame, Externally Generated Frame Pulse**

As with the T1 protocol, the TX frame sync can be generated externally. When dealing with the externally sourced frame sync signal, the logic on the data path within the HSS must be accounted for, thus an offset must be programmed within the HSS.

Figure 175. E1 RX Frame, Externally Generated Frame Pulse


By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for E1 operation:

- Frame size 256 bits (for E1).
- Frame sync simultaneous with first bit in first timeslot – TX frame offset and RX frame offset should be set due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to not use FBit at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

13.5.3 GCI

The HSS hardware has support to allow connection to a General Circuit Interface (GCI). This interface is also sometimes referred to as an IOM (ISDN Oriented Modular) interface. The HSS hardware supports two basic modes of operation for GCI protocol:

- Line-Card Mode
- Termination Mode

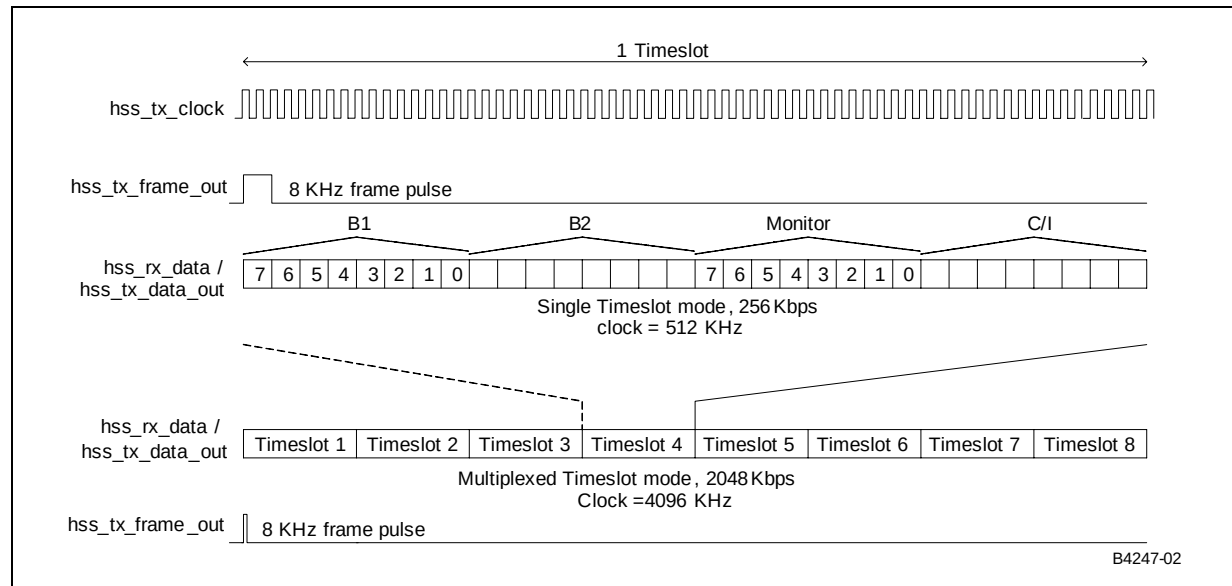
13.5.3.1 Line-Card Mode

The General Circuit Interface (also know as IOM) is a serial data stream consisting of 8000 frames, 4 slots per frame and each slot is a byte wide. It is used to exchange Bearer channel data (PCM-coded voice) and control information.



The HSS also supports GCI multiplexed mode, which means a number of GCI-compatible devices can be connected to the same serial bus, the numbers allowed are 1, 3, 4 and 8.

Figure 176. GCI Frames, Internally Generated Frame Pulse (Line-Card Mode)



When in multiplex mode and using 8 timeslots, GCI can support 8 ISDN transceivers (1 per timeslot).

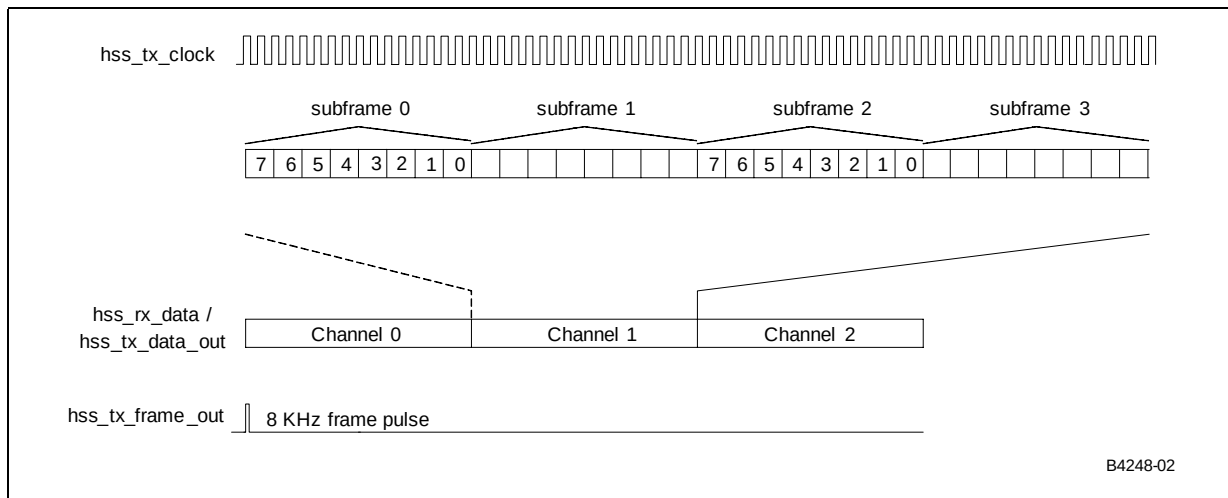
Figure 176 illustrates the frame structure used in GCI when in line-card mode.

Table 243. Timeslot Configurations

Timeslot Mode	Clock Frequency	Frame Size
1	512 KHz	32 bits
3	1.536 MHz	96 bits
4	2.048 MHz	128 bits
8	4.096 MHz	256 bits

13.5.3.2 Termination Mode

This mode is a variation of Line-card mode. The frame consists of 12 sub-frames, each containing 8 bits. Each 12 byte frame is repeated at 8 KHz, giving an aggregate data rate of 768 Kbps.

Figure 177. GCI Frames, Internally Generated Frame Pulse (Termination Mode)


The clock runs at 0.768 MHz, unlike other GCI protocols the clock rate is equal the data rate.

The following settings should be used when programming the HSS for GCI,

- Set frame size for GCI.
- Frame sync simultaneous with first bit – set TX frame offset and RX frame offset set to due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to not use FBit in the frame.
- Select value for idle timeslots on transmit.
- Configure buffer size.

13.5.4 MVIP

MVIP provides a method of interlacing E1 streams onto a single E1 line, and multiple T1 streams onto a single T1 line. Increasing the clock speed and alternating various timeslots helps provide this functionality. A single T1 line can also be mapped into an E1 line.

The NPE Core can program the HSS in the following ways.

Consider the following when programming the HSS for MVIP:

- Set frame size for MVIP.



- Frame sync simultaneous with first data nibble – set TX frame offset and RX frame offset due to HSS logic, different values due to external device can be accommodated.
- Select use of input/output TX/RX frame syncs.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame sync active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select not to use FBit in the frame.
- Select level for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Set interlace mode (byte/frame)
- Set lookup tables.

The clocks must be capable of running at 1.544 MHz, 2,048 MHz, 4.096 and 8.192 MHz for MVIP mode.

The following three sections describe the three ways in which the protocol can operate. The TX and RX are not explicitly described below, as the RX side of the protocol is identical to the TX side of the protocol.

13.5.4.1 2.048-Mbps Backplane

The clock rate for this form of T1 is 2.048Mbps as opposed to the usual T1 rate of 1.544 Mbps. The data rate remains at 1.544Mbps, meaning that certain timeslots within the frame carry no data and thus are discarded by the HSS (unassigned timeslots, configured by the look up tables). Figure 178 below illustrates the point.

Figure 178. MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame

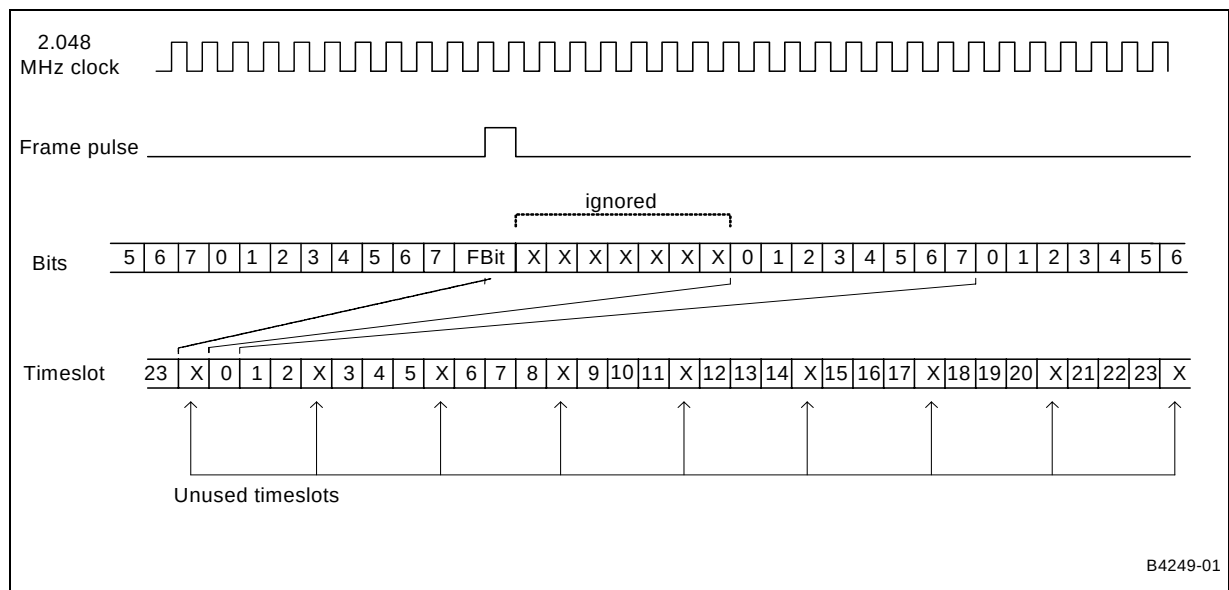




Figure 178. MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame

Note: In timeslot 0, the frame timeslot is not ignored

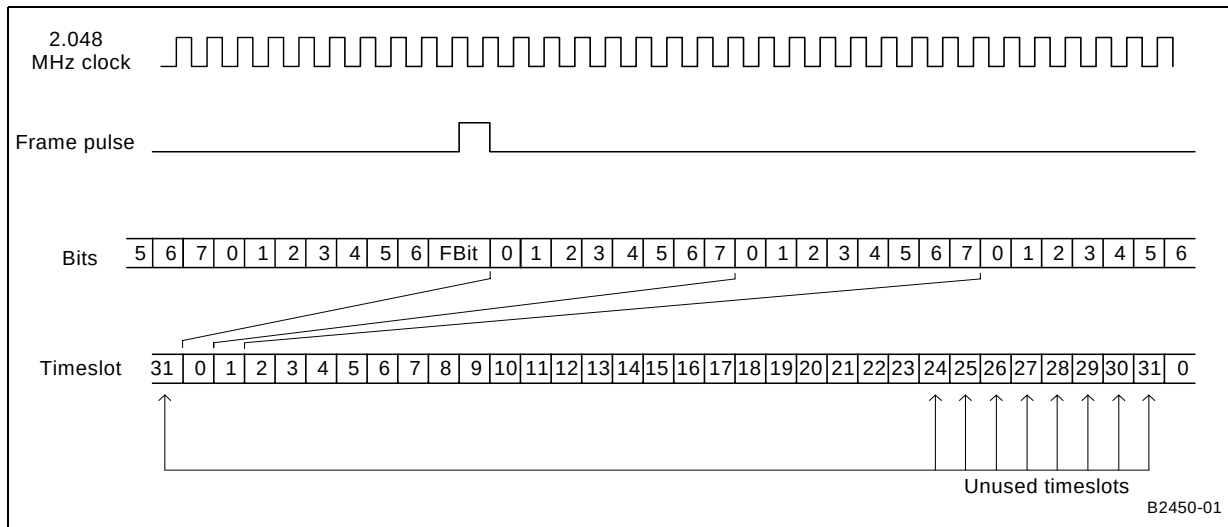
Every fourth timeslot received by the HSS is discarded, meaning it is not loaded into the FIFO and is therefore not sent to the NPE Core.

The HSS can transmit all zeros/ones (NPE Core programmable) for the duration of the unassigned timeslots.

Received unassigned timeslots are not sent to the NPE Core as they are discarded by the HSS.

Another method of sending a T1 line over an E1 line called frame mapping is illustrated in Figure 179. This method groups the 24 timeslots together and places unassigned timeslots towards the end of the frame. The FBit is located at the last bit of the 32nd timeslot, the HSS will not treat this timeslot any differently to other timeslots, it is up to the software to detect the FBit. The NPE Core can select which method it desires to use when implemented in NPE software.

Figure 179. MVIP, Frame Mapping a T1 Frame to an E1 Frame

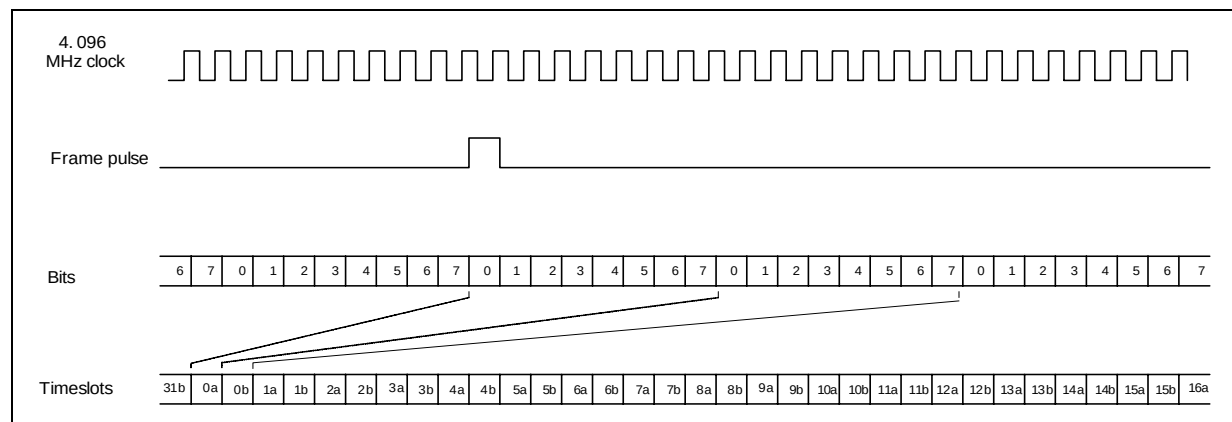


The NPE Core can program the HSS to automatically ignore (lookup table assigned) the last eight timeslots. Meaning the NPE Core will not receive the contents of the last eight timeslots. When timeslot 23 is transmitted, the next data from the NPE Core will not be transmitted until timeslot zero occurs. The HSS will transmit all zeros/ones for the duration of the empty timeslots (NPE Core programmable).

The NPE Core must program the HSS to indicate which method of T1 mapping is used (if any).

13.5.4.2 4.096-Mbps Backplane

This backplane is used to transport two E1s or two T1s on a single line which allows a clock rate of 4.096 MHz. Two complete E1 frames will fill this frame, therefore unassigned timeslots are not compulsory. When transporting T1 frames, unassigned timeslots are used for padding the frame due to the shorter length of the T1 frame.


Figure 180. MVIP, Byte Interlacing Two E1 Streams onto a 4.096-Mbps Backplane


In [Figure 180](#), the 'a' denotes the first E1 stream, the 'b' denotes the second E1 stream, the two streams are interlaced byte wise. Another method of placing E1 stream on this backplane is to process the first entire E1 stream followed by the second complete E2 stream (frame interleaving).

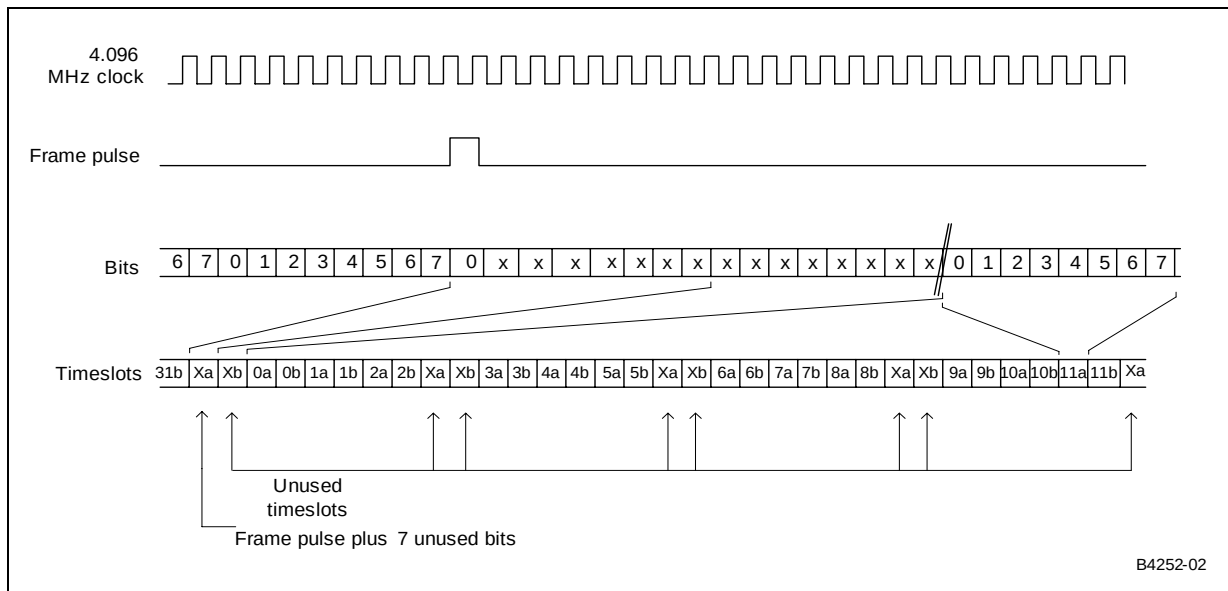
[Figure 181](#) illustrates a method in which two T1 frames are placed on a 4.096-Mbps backplane. It can be seen that the first two timeslots are unassigned with the exception of the frame pulse. The first three timeslots of each T1 stream are then placed (interlaced) in succession on the bus, then one unassigned timeslot per T1 stream present are placed on the bus. Unassigned RX bytes do not pass through the HSS FIFO (lookup tables give unassigned timeslots).

The HSS can also transmit unassigned timeslots, the value of which is programmable. The NPE Core need only supply the contents of the T1 frames, it does not need to transmit unused timeslots to the HSS. The location of these unassigned timeslots are defined by the lookup table.

The backplane can contain the 2 T1 streams byte interlaced as shown in [Figure 181](#) or the T1 stream can be placed in its entirety first followed by eight unassigned timeslots (frame pulse at the last bit of the 32nd timeslot. The second T1 stream then commences followed by eight unassigned timeslots. The frame pulse is coincidental with the last bit in the 32nd timeslot. The second timeslot follows the format of the first timeslot and together take up the 64 timeslots available. Once again the HSS can be programmed by the NPE Core to ignore the eight unassigned timeslots while taking the frame bit into account.



Figure 181. MVIP, Byte Interleaving Two T1 Streams onto a 4.096-Mbps Backplane

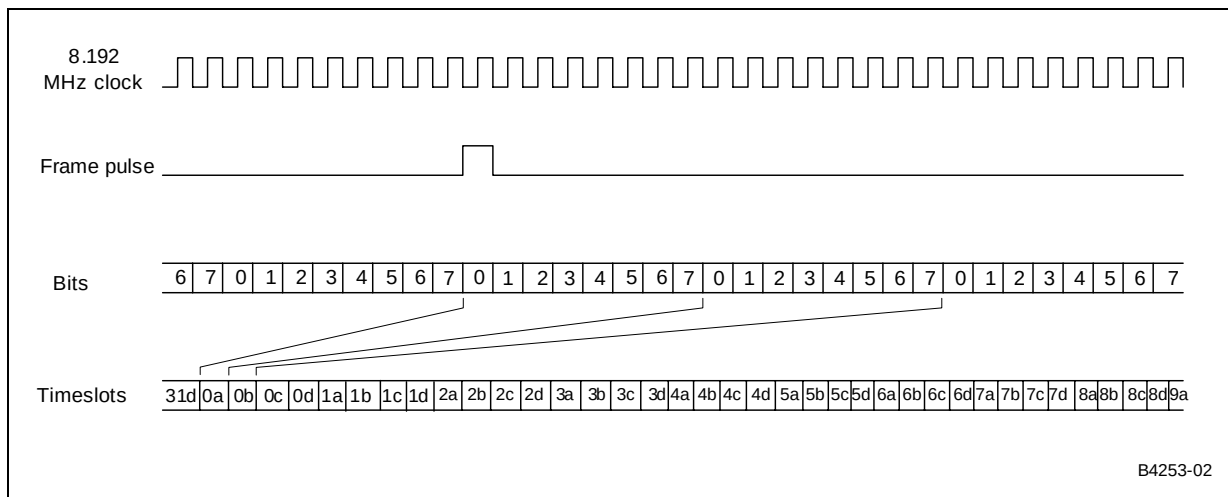


13.5.4.3 8.192-Mbps Backplane

This backplane supports 4 E1 or 4 T1 streams on a single line which require that the interface operates at 8.192 MHz.

Figure 182 illustrates that 4 E1 streams can be byte interleaved. The frame pulse occurs at the first byte of the first E1 stream. No unassigned timeslots are necessary here as the E1 frames can completely fill all the timeslots available.

Figure 182. MVIP, Byte Interleaving Four E1 Streams onto a 8.192-Mbps Backplane Bus



When T1 frames are placed on this backplane bus, then unassigned timeslots are required as a T1 frame is 24 timeslots wide unlike E1, which is 32 timeslots wide.



Byte interleaving involves disregarding four in every 16 timeslots (as shown in [Figure 182](#)). The first four timeslots are unassigned except for the framing pulse. The following 4 bytes are byte 0 of each T1 frame. The next 4 bytes are byte 1 of each T1 frame. The next 4 bytes are byte 2 of each T1 frame. The following 4 bytes are again unassigned. Unassigned timeslots are dictated by the look up tables.

Frame interleaving T1 frames onto this backplane bus would be to process the first T1 frame in its entirety starting with the first timeslot and finishing on the 24th timeslot, the frame bit/pulse is located on the last bit of the 32nd timeslot.

The second T1 frame is then processed and so on until all 4 frames are processed, this fills the entire 128 timeslots available. This mode is also programmable by the NPE Core.





14.0 Universal Asynchronous Receiver-Transmitter (UART)

14.1 Overview

The Intel® IXP45X and Intel® IXP46X Product Line of Network Processors supports two Universal Asynchronous Receiver-Transmitter (UART) interfaces. One UART is intended primarily to support debug and control functions. The other UART is primarily intended to support connection to high-speed serial devices, such as Bluetooth.

Each UART supports four pins:

- Transmit Data
- Receive Data
- Request to Send
- Clear to Send

The UARTs perform serial-to-parallel conversion — on data characters received from a peripheral device or a modem — and parallel-to-serial conversion — on data characters received from the Intel XScale processor.

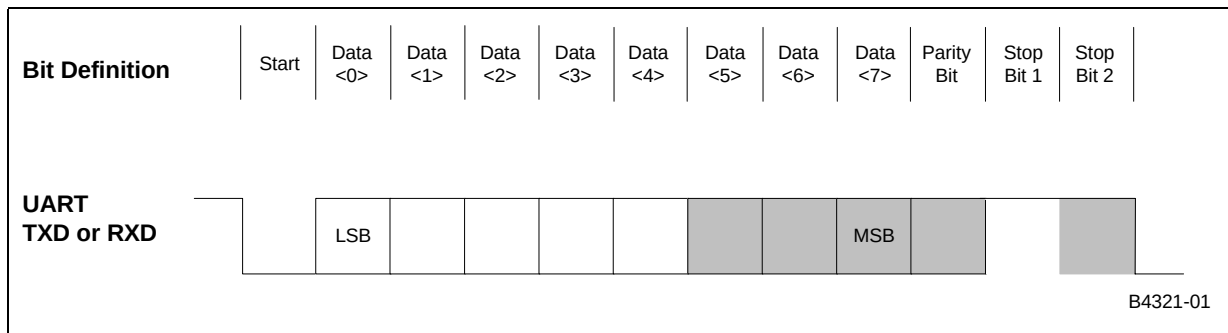
The Intel XScale processor, within the IXP45X/IXP46X network processors, can read the complete status of the UART at any time during functional operation. Available status information includes the type and condition of the transfer operations being performed by the UART, as well as any detected error conditions (such as parity, overrun, framing, or break interrupt).

The UART has enhancements in place to support higher speeds than defined by the 16550 UART specification. The UARTs are capable of supporting data transfers containing five, six, seven, or eight data bits. The data transfers may be configured to have one or two stop bits and supports even, odd, or no parity.

Figure 183 shows a functional waveform of the data that could be contained on the UART transmit and receive lines. Notice that Data bits 5 through 7, the Parity Bit, and Stop Bit 2 are shaded. The Data bits 5 through 7, Parity Bit, and Stop Bit 2 are all programmable and optional as previously described.



Figure 183. UART Timing Diagram



The serial port can operate in either FIFO or non-FIFO mode. In FIFO mode, a 64-byte transmit FIFO holds data coming from the Intel XScale processor to be transmitted on the serial link, and the 64-byte Receive FIFO, buffers data received from the serial link until the data is read by the Intel XScale processor. In Non-FIFO mode, data will be transmitted and received using two registers - the Transmit-Holding Register and the Receive-Buffer Register - along with the UART control, status and interrupt registers.

The UARTs include a programmable baud rate generator capable of dividing the 14.7456-MHz, UART input clock by divisors of 1 to ($2^{16} - 1$) and produces a 16X clock to drive the internal transmitter and receiver logic. The 14.7456-MHz, UART input clock is generated internally to the IXP45X/IXP46X network processors.

Interrupts can be programmed to the user's requirements, minimizing the computing required to handle the communications link. Each UART can be operated in a polled or an interrupt driven environment as selected by software.

The maximum baud rate supported by the UART is 921.6 Kbps. The divisors programmed in divisor latch registers should be equal to or greater than 1 for proper operation. The device UARTs may be initialized by setting 13 configuration registers.

14.2 Feature List

- Microprocessor Interface Control via Advanced Peripheral Bus (APB)
- Adds or deletes standard asynchronous communications bits (start, stop, and parity) to or from the serial data
- Independently controlled transmit, receive, line status and data set interrupts
- Programmable baud rate generator allows division of clock by 1 to ($2^{16} - 1$) and generates an internal 16X clock
- Modem control functions (cts_n and rts_n)
- Fully programmable serial-interface characteristics:
 - 5, 6, 7, or 8-bit characters
 - Even, odd, or no parity detection
 - 1, 1-1/2, or 2 stop bit generation
- Baud rate generation (up to 921kbps)
- False start bit detection
- 64-byte Transmit FIFO
- 64-byte Receive FIFO
- Complete status reporting capability



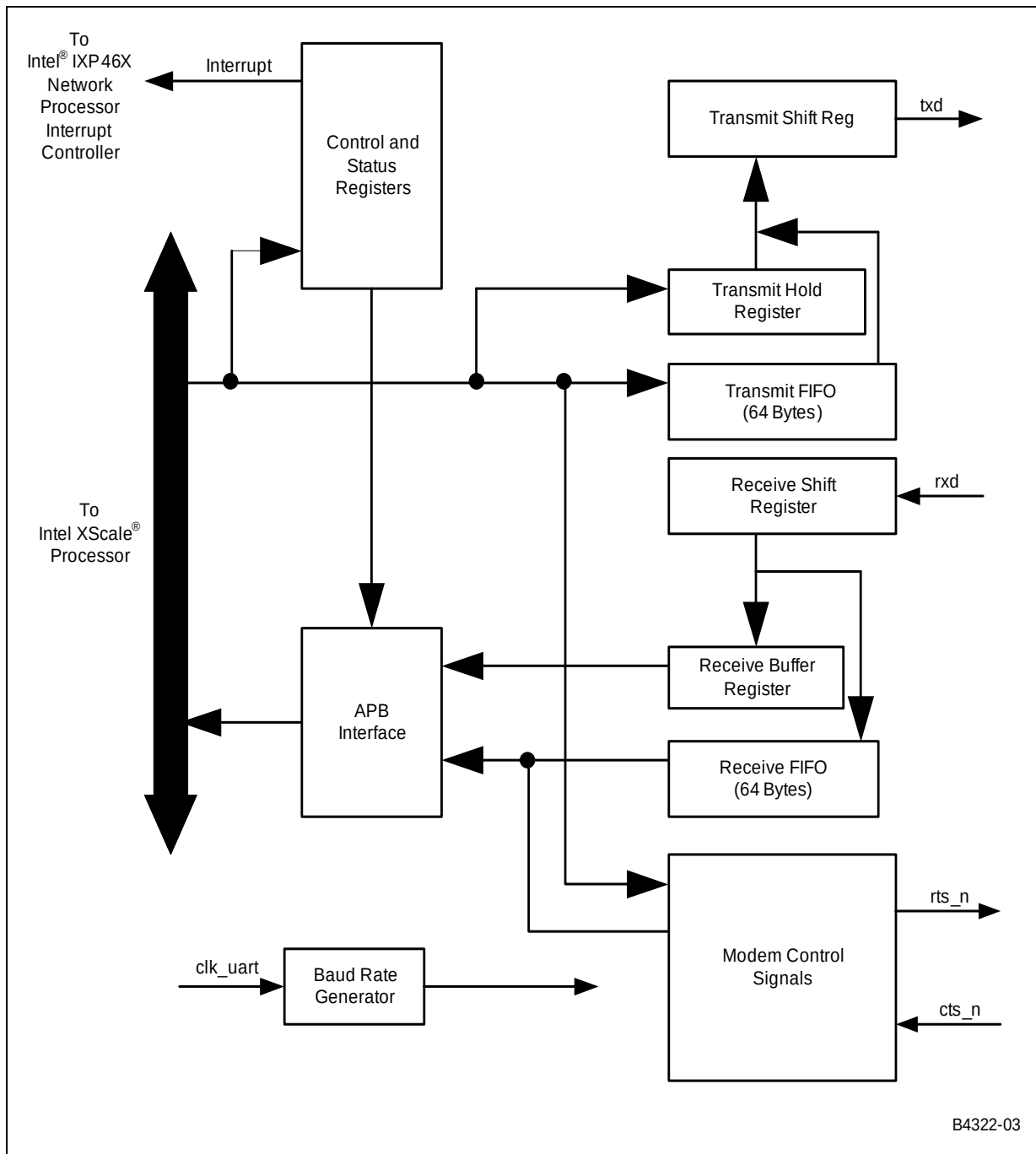
- Line break generation and detection
- Internal diagnostic capabilities include:
 - Loopback controls for communications link fault isolation
 - Break, parity, overrun, and framing error simulation
 - Fully prioritized interrupt system controls
- Two Data Service requests for transmit and receive data service

14.3 Block Diagram

Figure 184 shows a functional block diagram of the UART interface.



Figure 184. UART Block Diagram



14.4 Theory of Operation

The following sections provide a detailed description of configuring the UART interfaces for operation.



14.4.1 Setting the Baud Rate

Each UART contains a programmable baud-rate generator that is capable of taking the 14.7456 MHz, input clock (clk_uart) and dividing it by any divisor ranging from 1 to ($2^{16} - 1$). The output frequency of the baud-rate generator is 16 times the baud rate. So, if a 1,200 Baud rate was required, the output frequency of the baud-rate generator would be 1,200 KHz * 16 = 19,200 KHz.

Two 8-bit registers store the divisor in a 16-bit binary format: the Divisor Latch Low Register (DLL) and the Divisor Latch High Register (DLH). The Divisor Latch Low register makes up the lower eight bits of the 16-bit divisor and the Divisor Latch High register makes up the upper eight bits of the 16-bit divisor.

The two Divisor Latch registers must be loaded during initialization to ensure proper operation of the baud-rate generator. If both Divisor Latches are loaded with 0, the 16X output clock is stopped. A Divisor value of 0 in the Divisor Latch Low Register is not allowed.

The reset value of the divisor is hexadecimal 0x0002. The value of hexadecimal 0x0002 implies a value of hexadecimal 0x00 in the Divisor Latch High Register and a value of hexadecimal 0x02 in the Divisor Latch Low Register. The Divisor Latch High Register and Divisor Latch Low Register can only be written after the DLAB bit (bit 7 of the Serial Line Control Register) is set to logic 1.

The baud rate of the UART transmit and receive data is given by:

$$\text{Baud Rate} = 14.7456 \text{ MHz} / (16 \times \text{Divisor})$$

Table 244 shows some commonly used baud rates.

Table 244. Typical Baud-Rate Settings

Divisor Latch High Register	Divisor Latch Low Register	Divisor		Baud Rate Generator Clock Output	Baud Rate
		Hexadecimal	Decimal		
0x00	0x01	0x0001	1	14.7456 MHz	921,600
0x00	0x02	0x0002	2	7.3728 MHz	460,800
0x00	0x04	0x0004	4	3.6864 MHz	230,400
0x00	0x08	0x0008	8	1.8432 MHz	115,200
0x00	0x10	0x0010	16	921.6 KHz	57,600
0x00	0x20	0x0020	32	460.8 KHz	28,800
0x00	0x30	0x0030	48	307.2 KHz	19,200
0x00	0x40	0x0040	64	230.4 KHz	14,400
0x00	0x80	0x0080	96	115.2 KHz	9,600
0x00	0xC0	0x00C0	192	76.8 KHz	4,800
0x01	0x80	0x0180	384	38.4 KHz	2,400
0x03	0x00	0x0300	768	19.2 KHz	1,200

14.4.2 Setting Data Bits/Stop Bits/Parity

The Line Control Register (LCR) is an 8-bit register that enables the system programmer to specify the format of the asynchronous data communications exchange. The serial data format consists of a start bit (logic 0), five to eight data bits, an optional



parity bit, and one or two stop bits (logic 1). The Line Control Register also contains a bit used for accessing the Divisor Latch Registers and causing a UART break condition. The programmer also has the ability to read the contents of the Line Control Register.

The 8-bit Line Control Register is broken up into seven smaller registers

- Divisor-Latch Access Bit
- Set-Break Bit
- Sticky-Parity Bit
- Even-Parity Select Bit
- Parity-Enable Bit
- Stop-Bits Bit
- Word-Length Select Bits

The Divisor-Latch Access Bit must be set to logic 1 to access the Divisor Latch Registers of the baud-rate generator. When the Divisor-Latch Access Bit is set to logic 0, accessing the same addresses as defined for accessing the Divisor Latch Registers will allow access to the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.

The Set-Break Control Bit causes a UART break condition to be transmitted to the receiving UART. When Set-Break Control Bit is set to logic 1, the serial output data (TXD) is forced to the spacing or logic 0 state and remains in the spacing state until Set-Break Control Bit is set to logic 0. When the Set-Break Control Bit is set to logic 0, the serial output data is transmitted normally from the transmit-holding register. The Set-Break feature enables the processor to alert a terminal in a computer communications system.

The Sticky-Parity Bit (STKYP) is used to transmit the complement of the Even-Parity Select when parity is enabled. When the Parity-Enable Bit and Sticky-Parity Bit are logic 1, the bit that is transmitted — in the parity bit location of the serial output data stream — is the complement of the Even-Parity Select Bit.

For example, the Even-Parity Select Bit is logic 0 and the parity-bit location of the serial output data stream will be transmitted as logic 1. When the Sticky-Parity Bit and Parity-Enable bits are logic 1, the receiver logic compares the parity bit that is received — in the parity bit location of the serial data input stream — with the complement of the Even-Parity Select Bit.

If the values being compared are not equal, the receiver sets the Parity-Error Bit in Line-Status Register and causes an interrupt error if Line-Status Interrupts are enabled.

If the Even-Parity Select Bit is logic 0, the receiver expects the bit received at the parity-bit location of the serial data input stream to be logic 1. If the parity bit received is logic 0, the parity-error bit is set to logic 1. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as UART data.

If Parity enable is set to logic 0, Sticky Parity will be ignored.

The Even-Parity Select (EPS) Bit is used to determine the parity type to transmit or check on receive data when the Parity-Enable (PEN) Bit in the Line-Control Register enables parity. When the Parity-Enable bit is logic 1 and the Even-Parity Select bit is logic 0, the parity generator will transmit odd parity and the parity checker will validate against odd parity on the received data.

When the Parity-Enable Bit is logic 1 and the Even-Parity Select Bit is logic 1, the parity generator will transmit even parity and the parity checker will validate against even parity on the received data.



Table 245 shows the serial output data configurations for transmission of the parity bit.

Table 245. UART Transmit Parity Operation

PEN	EPS	Data to be Transmitted (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit to be Transmitted
1	0	10101010	1
1	0	10101011	0
1	1	10101010	0
1	1	10101011	1
0	X	XXXXXXXX	No Parity Bit Sent

Table 246 shows the serial input data configurations for reception of data containing a parity bit.

Table 246. UART Receive Parity Operation

PEN	EPS	Data being Received + Parity Bit (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit Should Be
1	0	101010101	1
1	0	101010110	0
1	1	101010100	0
1	1	101010111	1
0	X	XXXXXXXX	Parity Checking Disabled

The Stop-Bits Bit (STB) configures the number of stop bits to be transmitted or received in each serial character. When the Stop-Bits Bit is logic 0, one stop bit is generated in the transmitted data. When the Stop-Bits Bit is logic 1, and a 5-bit word length is selected — via the Word Length select bits in the Line Control Register — 1.5 stop bits will be transmitted.

When the Stop-Bits Bit is logic 1 and the word length is selected as a 6, 7, or 8-bit word, 2 stop bits are transmitted. The UART receiver logic checks the first stop bit only, regardless of the number of stop bits configured by the Stop Bits bit.

The Word-Length Select (WLS) Bits specify the number of data bits contained in each transmitted or received serial character. The Word-Length Select Bits configuration is shown in Table 247.

Table 247. UART Word-Length Select Configuration

WLS		Number of Data Bits Contained in Each Transmitted Or Received Character
Bit 1	Bit 0	
0	0	5-bit character (default)
0	1	6-bit character
1	0	7-bit character
1	1	8-bit character

The Line-Control Register is initialized to hexadecimal 0x00 after reset. The Line-Status Register is initialized to hexadecimal 0x60 after reset.



14.4.3 Using the Modem Control Signals

The IXP45X/IXP46X network processors provide two modem control signals, the Clear-to-Send input (CTS_N) and the Ready to Send output (RTS_N).

The Clear-to-Send input signal is sampled by reading the Modem Status Register and the Ready-to-Send output is controlled by the Modem Control Register. The Modem Control Register is a 5-bit register that provides control for four modem control signals:

- OUT1
- Ready-to-Send
- OUT2
- Loop-back test control bit
- Data Terminal Ready

Only one of the four modem-control signals (Ready-to-Send) is replicated to the external pins of the IXP45X/IXP46X network processors. The three other modem-control signals are only utilized when the UART is being used in loop-back diagnostic mode. Bit 4 of the Modem-Control Register is the Loop Back Test Mode bit (LOOP). The Loop Back Test Mode bit provides a local loop-back feature for diagnostic testing of the UART.

When the Loop Back Test Mode bit is set to logic 1, the following event will occur:

- The transmitter serial output is set to logic-1 state.
 - The receiver serial input is disconnected from the pin.
 - The output of the Transmitter Shift Register is "looped back" into the Receiver Shift Register input.
 - The Clear-to-Send input signal is disconnected from the pin
 - The Ready-to-Send output signal is forced to logic 1.
 - The lower four bits of the Modem Control Register are connected to the upper four bits of the Modem Status Register
- Status register bit mapping, while in loop back mode:
- DTR = 1 forces DSR to a 1
 - RTS = 1 forces CTS to a 1
 - OUT1 = 1 forces RI to a 1
 - OUT2 = 1 forces DCD to a 1

Leaving loop-back mode and returning to normal mode may result in unpredictable activation of the Modem Status Register (MSR). It is recommended that the Modem Status Register be read once to clear the Modem Status Bits in the Modem Status Register.

In the loop-back diagnostic mode, data that is transmitted will be immediately received. The ability to loop back the transmit data path to the receive path allows the Intel XScale processor to verify the transmit data path and receive data path of the UART. The transmit interrupts, receive interrupts, and modem-control interrupts are operational, when placed in the loop-back diagnostic mode. The Modem-Control Register Bits — instead of the Modem Control inputs — will activate the modem-control interrupts. A break signal can also be transferred — from the transmitter section to the receiver section — while operating in loop-back diagnostic mode.

Bit 1 of the Modem-Control Register is the Request-to-Send Bit. The Request-to-Send Control Bit is used to program the RTS_N output pin. When the Request-to-Send control bit is programmed to logic 0, the RTS_N signal will output logic 1. When the Request-to-Send Control Bit is programmed to logic 1, the RTS_N signal will output logic 0.



The Modem-Status Register is used to monitor the status of an external modem or data set. The Modem Status Register is an 8-bit register that is used to detect when a modem is capable of accepting new data. The status of the modem to accept incoming data is monitored by reading the Clear-to-Send Bit of the Modem Status Register and the Delta Clear-to-Send Bit of the Modem Status Register. The other six bits of the Modem-Status Register can be used for UART debug purposed only.

Modem Status Register Bit 4 is the Clear-to-Send (CTS). The Clear-to-Send Bit will be the complement of the Clear-to-Send (CTS_N) input signal. The Clear-to-Send signal will be connected to the Ready-to-Send bit of the Modem Control Register when the LOOP bit — in the Modem Control Register — is set to logic 1.

- CTS = logic 0 = CTS_N pin is 1
- CTS = logic 1 = CTS_N pin is 0

Modem Status Register Bit 0 is the Delta Clear-to-Send (DCTS). The Delta Clear-to-Send bit will inform the IXP45X/IXP46X network processors that nothing has happened to the Clear-to-Send Status since the last time that the Modem-Status Register was read. The Delta Clear-to-Send bit will be set to logic 0 after a read of the Modem-Status Register.

- DCTS = logic 0 = No change in CTS_N pin since last read of the Modem-Status Register
- DCTS = logic 1 = CTS_N pin has changed state since the last read of the Modem-Status Register

The Modem-Control Register is initialized to hexadecimal 0x00 after reset. The Modem-Status Register is initialized to hexadecimal 0x00 after reset.

14.4.4 UART Interrupts

The UART Interrupt Enable Register (IER) is an 8-bit register that enables five types of UART based interrupts and enables the UART functionality and other control functionality not used by the IXP45X/IXP46X network processors.

UART Interrupt Enable Register bit 6 is the UART Unit Enable (UUE) bit. When the UART Unit Enable bit is set to logic 0, the UART will be completely non-functional. Likewise, when the UART Unit Enable bit is set to logic 1, the UART will be enabled.

UART Interrupt Enable Register bits 4 through 0 represent five different interrupt types that can be individually enabled/disabled:

- Receiver Time Out Interrupt Enable (RTOIE)
- Modem Interrupt Enable (MIE)
- Receiver Line Status Interrupt Enable (RLSE)
- Transmit Data Request Interrupt Enable (TIE)
- Receiver Data Available Interrupt Enable (RAVIE)

The Receiver Line Status Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors and captured in the UART Interrupt Identification Register (IIR) when a receive error is detected. Such Receiver Line Status Conditions that would cause the interrupt to occur are:

- Overrun
- Parity
- Framing
- Break
- FIFO error



In FIFO mode, each received character carries the line status along with the character. When in FIFO Mode, the Receive Line Status Interrupt will be active on the character located in the bottom of the Receive FIFO. If a Line Status error condition is detected on the character in the bottom of the Receive FIFO, a Receive Line Status Interrupt is generated. Reading the Line Status Register clears the Receive Line Status Interrupt.

When in Non-FIFO Mode, the Receive Line Status Interrupt will be active on the character located in the bottom of the Receive Buffer Register. If a line-status error condition is detected on the character in the Receive Buffer Register, a Receive Line Status Interrupt is generated.

The Receiver Data Available Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors and captured in the UART Interrupt Identification Register (IIR) when UART data is available to be read by the Intel XScale processor.

When the UART is in FIFO mode, the Receive Data Available interrupt will be encoded in the Interrupt Identification Register, after the FIFO trigger level defined in the FIFO Control Register (FCR) is reached. When the UART is in non-FIFO mode, the Receive Data Available interrupt will be encoded in the Interrupt Identification Register after data is in the Receive Buffer Register (RBR).

When operating in FIFO Mode, the Receive Data Available Interrupt is cleared, when the FIFO drops below the programmed trigger level. When operating in Non-FIFO Mode, the Receive Data Available Interrupt is cleared when the received character is read by the IXP45X/IXP46X network processors from the Receive Holding Register.

The Receiver Interrupt Time Out Enable can be used only in FIFO Mode and allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors and captured in the UART Interrupt Identification Register (IIR). This happens when:

- At least one character is available in the receive FIFO
- The last character received by the UART receive interface occurred more than four continuous character times ago
- The most-recent read of the receive FIFO by the IXP45X/IXP46X network processors was more than four continuous character times ago

For example, the maximum time between a received character and a Receive Character Time-Out Interrupt is 160 ms at 1,200 baud with a 12-bit receive character (i.e., 1 start, 8 data, 1 parity, and 2 stop bits).

$$\begin{aligned} & (1 / ((1200 \text{ characters/second}) / 12 \text{ characters})) * 4 \\ & \text{characters} = \\ & = 0.040 \text{ seconds} \\ & = 40 \text{ ms} \end{aligned}$$

The time-out interrupt is cleared by the IXP45X/IXP46X network processors reading the Receive FIFO or setting the RESETRF bit to logic 1, in the FIFO Control Register.

If a Receive Character Time-Out Interrupt is active, the interrupt-time-out counter will be reset only after the IXP45X/IXP46X network processors read a character from the Receive FIFO. If a Receive Character Time-Out Interrupt is non-active, the interrupt time-out counter will be reset after the IXP45X/IXP46X network processors read a character from the Receive FIFO or when a new character is placed into the Receive FIFO by the receive interface.

The Transmit Data Request Interrupt Enable can be used in Non-FIFO mode or FIFO Mode.



In Non-FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors and captured in the UART Interrupt Identification Register (IIR), when the Transmit Holding Register is empty. Reading the Interrupt Identification Register or writing a new character into the Transmit Holding Register clears the Transmit Data Request Interrupt.

In FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors and captured in the UART Interrupt Identification Register (IIR), when the Transmit FIFO is half empty or less.

The Transmit FIFO size is 64 characters. When 32 or fewer characters are remaining in the Transmit FIFO to be transmitted, the Transmit Data Request Interrupt will be generated. Reading the Interrupt Identification Register — or writing a new data into the Transmit FIFO — clears the Transmit Data Request Interrupt.

The Modem Status Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP45X/IXP46X network processors — and captured in the UART Interrupt Identification Register (IIR) — when the Clear-to-Send, Data-Set-Ready, Ring-Indicator, or Received-Line Signal Detect bits in the Modem Status Register are set to logic 1. Reading the Modem Status Register clears the Modem Status Interrupt.

Clearing the appropriate bit of the Interrupt Enable Register may disable each of the interrupt types previously described. Similarly, by setting the appropriate bits, selected interrupts can be enabled on a per-interrupt basis.

When the UART interrupts are disabled, the UART is placed into polled mode of operation. Since the UART receiver and the UART transmitter are controlled separately, either one or both interfaces can be placed in the polled mode of operation.

In the polled mode of operation, software routines running on the Intel XScale processor checks receiver and transmitter status via the Line Status Register. Line Status Register bit 0 will be logic 1, when a character is available to be read from the Receive Interface. Lines Status Register bits 1 through 4 specify which error(s) has occurred — for the character at the bottom of the FIFO or in the Receive Buffer Register.

In FIFO mode, Line Status Register bit 1 through 3 is stored with each received character in the Receive FIFO. The Line Status Register shows the status bits of the character at the bottom of the Receive FIFO. When the character at the bottom of the FIFO has errors, the Line Status error bits are set and are not cleared until the Intel XScale processor reads the Line Status Register. Even if the character in the FIFO is read — and a new character is now at the bottom of the FIFO — the interrupts will not be cleared until the Line-Status Register is read.

Character-error status is handled in the same way as when the UART is operating in interrupt mode of operation. Setting Line-Status Register bit 5 to logic 1 indicates that the Transmit FIFO or the Transmit Holding Register is requesting data. Line Status Register bit 6 identifies that both the Transmit FIFO and the Transmit Shift Register have no data. Line Status Register bit 7 indicates the status of any errors in the Receive FIFO.

In non-FIFO mode, three of the LSR register bits — parity error, framing error, and break interrupt — show the error status of the character that has just been received.

The Receive Time-Out Interrupt is separated from the Receive-Data-Available Interrupt to prevent an Interrupt Controller routine and a Data Service controller routing from servicing the receive FIFO at the same time. Bit 7 of the Interrupt Enable Register is used as the enable bit of Data Service requests. Bit 5 of the Interrupt Enable Register is used as the enable bit of NRZ-coding enable.



Bits 7 and 5 are not implemented by the IXP45X/IXP46X network processors. The use of bit 7 through bit 4, of the Interrupt Enable Register, is defined differently from the register definition of standard 16550 UART.

The Interrupt Enable Register is initialized to all zeros after receiving a reset. The Interrupt Identification Register is a hexadecimal 0x01. Bit 5 and Bit 4 of the Interrupt Identification Register will always be logic 0.

14.4.5 Transmitting and Receiving UART Data

Transmitting and receiving data using the UARTs for the IXP45X/IXP46X network processors is accomplished in two modes: Non-FIFO Mode and FIFO Mode.

In Non-FIFO mode, data will be transmitted and received using two registers — the Transmit-Holding Register (THR) and the Receive-Buffer Register (RBR) — along with the UART control, status, and interrupt registers.

In FIFO mode, data will be transmitted and received using two 64-entry FIFOs, the Transmit FIFO, and the Receive FIFO, along with the UART Control, Status, and Interrupt registers.

The Transmit FIFO is 64 entries deep by 8 bits wide. The Transmit FIFO sizing allows a complete 8-bit data character to be stored in each entry. When characters smaller than 8 bits are transmitted, they are right-justified.

If a 5-bit character is to be transmitted, the character is represented by a binary 10110. The value located in the FIFO entry will be hexadecimal 0x16.

The Receive FIFO is 64 entries deep and 11 bits wide. The Receive FIFO sizing allows for an 8-bit character to be received along with the over-run flag, parity error flag, and framing error flag for each received character. Smaller characters will be right-justified, as described for the transmit FIFO.

The error flags position will remain constant, independent of the character size. The mode of operation and FIFO control parameters will be programmed using the FIFO Control Register (FCR).

The FIFO Control Register is an 8-bit register that configures the UARTs' mode of operation. The Transmit and Receive FIFO Enable Bit (TRFIFOE) — Bit 0 of the FIFO Control Register — determines the UARTs' mode of operation: FIFO Mode or Non-FIFO Mode. When set to logic 0, the UART will function in Non-FIFO Mode. When set to logic 1, the UART will function in FIFO Mode.

Two bits of the FIFO Control Register are used to reset the Transmit and Receive FIFO: the Reset Transmit FIFO bit (bit 2 of FCR) and the Reset Receive FIFO (bit 1 of FCR). Writing logic 0 to these bits has no effect. Writing logic 1 to the Reset Transmit FIFO bit will cause the Transmit FIFO counter to be reset to 0 and the transmit-data request bit to be set in the Line-Status Register.

Writing logic 1 to the Reset Receive FIFO bit will cause the Receive FIFO counter to be reset to 0 and the data ready bit in the Line-Status Register to be cleared. The Overrun Error Flag, Parity Error Flag, Framing Error Flag, and Break Interrupt Flag in the Line Status Register will remain unaltered. The Reset Transmit FIFO and Reset Receive FIFO will be cleared autonomously when the reset has been completed.

The Receive FIFO interrupt trigger level also is set in the FIFO Control Register. The Receive FIFO interrupt trigger level is used to generate an interrupt when the number of characters in the Receive FIFO is greater than to or equal to the trigger-level value. The Interrupt Trigger Level is defined as bit 6 and bit 7 of the FIFO Control Register. The bit definitions are shown in [Table 248](#).



Table 248. UART FIFO Trigger Level

Interrupt Trigger Level [7:6]	Description
00	1 byte or more in the FIFO causes an interrupt
01	8 bytes or more in the FIFO causes an interrupt
10	16 bytes or more in the FIFO causes an interrupt
11	32 bytes or more in the FIFO causes an interrupt

The UART must be configured prior to transmitting and receiving data to and from the UART. The Transmit Holding Register (THR) is used to transmit characters over the UART interface. The Receive Buffer Register is used to receive characters from the UART interface.

Transmitting UART data can be implemented using FIFO Mode or Non-FIFO mode. In FIFO mode, writing a character to the Transmit Holding Register will put data on the top of the transmit FIFO. In Non-FIFO mode, writing a character to the Transmit Holding Register will put data in the Transmit Holding Register. The next character transmitted will be the character contained in the Transmit Holding Register.

If characters less than 8 bits are sent, the characters will need to be right-justified. For example, if a 5-bit data character is to be transmitted with a binary value of 01011. The data written to the Transmit Holding Register will need to be written as hexadecimal 0x0B.

Receiving UART data can be implemented using FIFO Mode or Non-FIFO mode. In FIFO mode, reading a character from the Receive Buffer Register will read a character from the bottom of the receive FIFO. In Non-FIFO mode, reading a character from the Receive Buffer Register will read the data contained in the Receive Buffer Register. The next character received will be the character contained in the Receive Buffer Register.

If characters less than 8 bits are received, the characters will need to be right-justified. For example, if a 5-bit data character is received having a binary value of 00101. The data read from the Receive Buffer Register will be a hexadecimal 0x05. (Notice that the three most-significant bits of the byte are filled with zeros.)

The UART transmit data pin is logic 1 and the transmit FIFO and receive FIFO pointers are initialized to the empty value after reset.

14.5 Register Descriptions

There are 13 registers which monitor and control the UART. The registers are all 32 bits in size, but only the lower 8 bits have valid data. The UART registers share nine address locations in the I/O address space.

Note that the state of the Divisor Latch Bit (DLAB) — the most-significant bit of the Serial Line Control Register — affects the selection of certain of the UART registers. The DLAB bit must be set high by the system software to access the Baud Rate Generator Divisor Latches.

Table 249. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only



Table 249. Register Legend

Attribute	Legend	Attribute	Legend
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

The following sample register-summary table indicates, in parentheses, which paragraph tags are cross-referenced in the individual register tables.

Table 250. UART Registers Overview

Address	DLAB	Access	Name	Description
0xC800_X000†	0	RO	RBR	Receive Buffer Register
	0	WO	THR	Transmit Holding Register
	1	RW	DLL	Divisor Latch Low Register
0x C800_X004	0	RW	IER	Interrupt Enable Register
	1	RW	DLH	Divisor Latch High Register
0x C800_X008	0/1	RO	IIR	Interrupt Identification Register
	0/1	WO	FCR	FIFO Control Register
0x C800_X00C	0/1	RW	LCR	Line Control Register
0x C800_X010	0/1	RW	MCR	Modem Control Register
0x C800_X014	0/1	RO	LSR	Line Status Register
0x C800_X018	0/1	RO	MSR	Modem Status Register
0x C800_X01C	0/1	RW	SPR	Scratch-Pad Register
0x C800_X020	0	RW	ISR	Slow Infrared Select Register
† The X in the value C800_X000 is used to denote that this could be a value of either 0 or 1 depending upon if it is UART 0(High Speed UART) or UART 1(Console UART), respectively. This is the same for all other addresses to the UARTs.				

14.5.1 Receive Buffer Register

Register Name:		RBR																								
Hex Offset Address:		0xC800 X000								Reset Hex Value:		0x00000000														
Register Description:		Receive Buffer Register																								
Access: Read Only.																										
31																		8	7							0
(Reserved)																			RBR							



Register		RBR
Bits	Name	Description
31:8		(Reserved)
7:0	RBR	In non-FIFO mode, this register holds the character received by the UART's Receive Shift Register. If fewer than 8 bits are received, the bits are right-justified and the leading bits are zeroed. In FIFO mode, this register latches the value of the data byte at the bottom of the Receive FIFO. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

14.5.2 Transmit Holding Register

Register Name:		THR																										
Hex Offset Address:		0xC800 X000								Reset Hex Value:		0x00000000																
Register Description:		Transmit Holding Register																										
Access: Write Only.																												
31																		8	7									0
(Reserved)																				THR								

Register		THR
Bits	Name	Description
31:8		(Reserved)
7:0	THR	In Non-FIFO mode, this register holds the next data byte to be transmitted. When the Transmit Shift Register becomes empty, the contents of the Transmit Holding Register are loaded into the shift register and the transmit data request (TDRQ) bit in the Line Status Register is set to 1. In FIFO mode, writing to THR puts data to the top of the Transmit FIFO. The data at the bottom of the Transmit FIFO is loaded to the shift register when the shift register becomes empty. The DLAB bit in the Line Control Register must be set to logic 0 to access this register.

14.5.3 Divisor Latch Low Register

Register Name:	DLL																										
Hex Offset Address:	0xC800 X000								Reset Hex Value:	0x00000002																	
Register Description:	Divisor Latch Low Register																										
Access: Read/Write.																											
31																		8	7								0
(Reserved)																			DLL								



Register		DLL
Bits	Name	Description
31:8		(Reserved)
7:0	DLL	Lower byte of compare-value used by the baud rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

14.5.4 Divisor Latch High Register

Register Name:		DLH																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
Hex Offset Address:		0xC800 X004								Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
Register Description:		Divisor Latch High Register																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
Access: Read/Write.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
31																										8	7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												</

Register		DLH
Bits	Name	Description
31:8		(Reserved)
7:0	DLH	Upper byte of compare-value used by the baud-rate generator. The DLAB bit in the Line Control Register must be set to logic 1 to access this register.

14.5.5 Interrupt Enable Register

The DLAB bit in the Line-Control Register must be set to logic 0 to access this register.
DMA is not supported on the IXP45X/IXP46X network processors.

Register Name:		IER																								
Hex Offset Address:		0xC800 X004								Reset Hex Value:		0x00000000														
Register Description:		Interrupt Enable Register																								
Access: See below.																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																			DMAE	UUE	NRZE	RTOIE	RIE	RLSE	TIE	RAVIE



Register		IER
Bits	Name	Description
31:8		(Reserved)
7	DMAE	DMA Requests Enable: 0 = DMA requests are disabled 1 = DMA requests are enabled Not used on IXP45X/IXP46X network processors
6	UUE	UART Unit Enable: 0 = the unit is disabled 1 = the unit is enabled
5	NRZE	NRZ coding Enable: 0 = NRZ coding disabled 1 = NRZ coding enabled Not used on IXP45X/IXP46X network processors
4	RTOIE	Receiver Time Out Interrupt Enable: 0 = Receiver data Time out interrupt disabled 1 = Receiver data Time out interrupt enabled
3	RIE	Modem Interrupt Enable: 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled
2	RLSE	Receiver Line Status Interrupt Enable: 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled The DLAB bit in the Line Control Register must be set to logic 0 to access this register.
1	TIE	Transmit Data request Interrupt Enable: 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled
0	RAVIE	Receiver Data Available Interrupt Enable: 0 = Receiver Data Available (Trigger level reached) interrupt disabled 1 = Receiver Data Available (Trigger level reached) interrupt enabled

14.5.6 Interrupt Identification Register

In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the Interrupt-Identification Register. The Interrupt-Identification Register (IIR) stores information indicating that a prioritized interrupt is pending and the source of that interrupt.

Priority Level	Interrupt origin
1 (highest)	Receiver Line Status: One or more error bits were set.
2	Received Data is available: In FIFO mode, trigger level was reached. In non-FIFO mode, RBR has data.



Priority Level	Interrupt origin
2	Receiver Time out occurred: It happens in FIFO mode only, when there is data in the receive FIFO but no activity for a time period.
3	Transmitter requests data: In FIFO mode, the transmit FIFO is half or more than half empty. In non-FIFO mode, THR is read already.
4 (lowest)	Modem Status: One or more of the modem input signals has changed state.

Register Name:		IIR																														
Hex Offset Address:		0xC800 X008								Reset Hex Value:		0x00000001																				
Register Description:		Interrupt Identification Register																														
Access: See below.																																
31																								8	7	6	5	4	3	2	1	0
(Reserved)																							FIFOES		(Rsvd)		TOD	IID		IP_N		

Register		IIR
Bits	Name	Description
31:8		(Reserved)
7:6	FIFOES	FIFO Mode Enable Status: 00 = Non-FIFO mode is selected 01 = (Reserved) 10 = (Reserved) 11 = FIFO mode is selected (TRFIFOE = 1)
5:4		(Reserved)
3	TOD	Time Out Detected: 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)
2:1	IID	Interrupt Source Encoded: 00 = Modem Status (CTS, DSR, RI, DCD modem signals changed state) 01 = Transmit FIFO requests data 10 = Received Data Available 11 = Receive error (Overrun, parity, framing, break, FIFO error)
0	IP_N	Interrupt Pending: 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending



Table 251. UART IDD Bit Mapping

Interrupt ID Bits				Interrupt SET / RESET Function			
3	2	1	0	Priority	Type	Source	RESET Control
0	0	0	1	-	None	No Interrupt is pending	-
0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt	Reading the Line Status Register.
0	1	0	0	Second-Highest	Received Data Available.	Non-FIFO mode: Receive Buffer is full. FIFO mode: Trigger level was reached.	Non-FIFO mode: Reading the Receiver Buffer Register. FIFO mode: Reading bytes until Receiver FIFO drops below trigger level or setting RESETRF bit in FCR register.
1	1	0	0	Second-Highest	Character Time-out indication.	FIFO Mode only: At least 1 character is in receiver FIFO and there was no activity for a time period.	Reading the Receiver FIFO or setting RESETRF bit in FCR register.
0	0	1	0	Third-Highest	Transmit FIFO Data Request	Non-FIFO mode: Transmit Holding Register Empty FIFO mode: Transmit FIFO has half or less than half data.	Reading the IIR Register (if the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (if the source of the interrupt) or writing to the Transmitter FIFO.
0	0	0	0	Fourth-Highest	Modem Status	Clear-to-Send, Data Set Ready, Ring Indicator, Received Line Signal Detect	Reading the Modem Status Register.

14.5.7 FIFO Control Register

FCR is a write-only register that is located at the same address as the IIR. (IIR is a read-only register.) FCR enables/disables the transmitter/receiver FIFOs, clears the transmitter/receiver FIFOs, and sets the receiver FIFO trigger level.

Register Name:		FCR																								
Hex Offset Address:		0xC800 X008								Reset Hex Value:		0x00000000														
Register Description:		FIFO Control Register																								
Access: Write Only.																										
31																		8	7	6	5		3	2	1	0
(Reserved)																ITL		(Rsvd)		RESETRF	RESETRF	TRFIFOE				



Register		FCR
Bits	Name	Description
31:8		(Reserved)
7:6	ITL	Interrupt Trigger Level: When the number of entries in the receive FIFO equals the interrupt trigger level programmed into this field and the Received Data Available Interrupt is enabled (via IER), an interrupt is generated and appropriate bits are set in the IIR. 00 = 1 byte or more in FIFO causes interrupt 01 = 8 bytes or more in FIFO causes interrupt 10 = 16 bytes or more in FIFO causes interrupt 11 = 32 bytes or more in FIFO causes interrupt
5:3		(Reserved)
2	RESETTF	Reset Transmitter FIFO: When RESETTF is logic 1, the transmitted FIFO count logic is set to 0, effectively clearing all the entries in the transmit FIFO. The TDRQ bit in LSR are set and IIR shows a transmitter requests data interrupt if the TIE bit in the IER register is set. The transmitter shift register is not cleared; it completes the current transmission. After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0.
1	RESETRF	Reset Receiver FIFO: When RESETRF is set to 1, the receive FIFO counter is reset to 0, effectively clearing all the entries in the receive FIFO. The DR bit in the LSR is reset to 0. All the error bits in the FIFO and the FIFOE bit in LSR are cleared. Any error bits, OE, PE, FE or BI that had been set in LSR are still set. The receiver shift register is not cleared. If IIR had been set to Received Data Available, it is cleared. After the receive FIFO is cleared, RESETRF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	TRFIFOE	Transmit and Receive FIFO Enable: TRFIFOE enables/disables the transmitter and receiver FIFO s. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFO s are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFO s. 0 = FIFOs are disabled 1 = FIFOs are enabled

14.5.8 Line Control Register

Register Name:		LCR																									
Hex Offset Address:		0xC800 X00C								Reset Hex Value:		0x00000000															
Register Description:		Line Control Register																									
Access: Read/Write.																											
31																		8	7	6	5	4	3	2	1	0	
(Reserved)																				DLAB	SB	STKYP	EPS	PEN	STB		WLS



Register		LCR
Bits	Name	Description
31:8		(Reserved)
7	DLAB	Divisor Latch Access Bit: This bit must be set to logic 1 to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.
6	SB	Set break: This bit causes a break condition to be transmitted to the receiving UART. When SB is set to logic 1, the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is set to logic 0. This bit acts only on the TXD pin. 0 = No effect on TXD output 1 = Forces TXD output to 0 (space)
5	STKYP	Sticky Parity: This bit is the "sticky parity" bit, which can be used in multiprocessor communications. When PEN and STKYP are logic 1, the bit that is transmitted in the parity bit location (the bit just before the stop bit) is the complement of the EPS bit. If EPS is 0, then the bit at the parity bit location will be transmitted as a 1. In the receiver, if STKYP and PEN are logic 1, the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt if line status interrupts were enabled. For example, if EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, then the parity error bit is set. By forcing the bit value at the parity bit location — rather than calculating a parity value — a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, STKYP is ignored. 0 = No effect on parity bit 1 = Forces parity bit to be opposite of EPS bit value
4	EPS	Even-Parity Select: 0 = Sends or checks for odd parity 1 = Sends or checks for even parity
3	PEN	Parity enable: This is the parity enable bit. When PEN is logic 1, a parity character is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. 0 = No parity function 1 = Allows parity generation and checking
2	STB	Stop bits: This bit specifies the number of stop bits transmitted and received in each serial character. If STB is a logic 0, one stop bit is generated in the transmitted data. If STB is a logic 1 when a 5-bit word length is selected via bits 0 and 1, then 1 and one half stop bits are generated. If STB is a logic 1 when either a 6, 7, or 8-bit word is selected, then two stop bits are generated. The receiver checks the first stop bit only, regardless of the number of stop bits selected. 0 = One stop bit 1 = Two stop bits, except for 5-bit character, then 1.5 bits
1:0	WLS	Word-Length Select: Specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character (default) 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character



14.5.9 Modem Control Register

Register Name:		MCR																						
Hex Offset Address:		0xC800 X010								Reset Hex Value:		0x00000000												
Register Description:		Modem Control Register																						
Access: Read/Write.																								
31																			5	4	3	2	1	0
(Reserved)																				LOOP	OUT2	OUT1	RTS	DTR

Register		MCR (Sheet 1 of 2)
Bits	Name	Description
31:5		(Reserved)
4	LOOP	Loop back test mode: This bit provides a local Loop back feature for diagnostic testing of the UART. When LOOP is set to logic 1, the following will occur: - The transmitter serial output is set to a logic 1 state. - The receiver serial input is disconnected from the pin. - The output of the Transmitter Shift register is "looped back" into the receiver shift register input. - The four modem-control inputs (CTS_N, DSR_N, DCD_N, and RI_N) are disconnected from the pins and the modem control output pin RTS_N is forced to its inactive state. Note: Coming out of the loop back test mode may result in unpredictable activation of the delta bits (bits 3:0) in the Modem Status Register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. The lower four bits of the Modem Control register are connected to the upper four Modem Status register bits: - DTR = 1 forces DSR to a 1 - RTS = 1 forces CTS to a 1 - OUT1 = 1 forces RI to a 1 - OUT2= 1 forces DCD to a 1 — 0 = Normal UART operation — 1 = Test mode UART operation
3	OUT2	Interrupt Mask: The OUT2 bit is used to mask the UARTs' interrupt output to the Interrupt Controller unit. OUT2 = 0 UART interrupt masked OUT2 = 1 UART interrupt not masked When LOOP=1, this bit is not used as interrupt mask. The interrupt goes to the processor without mask in loop-back mode.



Register		MCR (Sheet 2 of 2)
Bits	Name	Description
2	OUT1	Test bit: This bit is used only in loop-back test mode. See LOOP row, above.
1	RTS	Request to Send: This bit controls the Request to Send (RTS_N) output pin. 0 = RTS_N pin is 1 1 = RTS_N pin is 0
0	DTR	Data Terminal Ready: 0 = DTR_N pin is 1 1 = DTR_N pin is 0

14.5.10 Line Status Register

Register Name:		LSR																								
Hex Offset Address:		0xC800 X014								Reset Hex Value:		0x00000060														
Register Description:		Line Status Register																								
Access: Read Only.																										
31																		8	7	6	5	4	3	2	1	0
(Reserved)																			FIFOE	TEMT	TDRQ	BI	FE	PE	OE	DR

Register		LSR (Sheet 1 of 2)
Bits	Name	Description
31:8		(Reserved)
7	FIFOE	FIFO Error Status: In non-FIFO mode, this bit is 0. In FIFO Mode, FIFOE is set to 1 when there is at least a parity error, framing error, or break indication for any of the characters in the FIFO. Note that a processor read to the Line Status register does not reset this bit. FIFOE is reset when all error bytes have been read from the FIFO. 0 = Non-FIFO mode or no errors in receiver FIFO 1 = At least one character in receiver FIFO has errors
6	TEMT	Transmitter Empty: TEMT is set to a logic 1 when the Transmit Holding register and the Transmitter Shift register are both empty. It is reset to logic 0 when either the Transmit Holding register or the transmitter shift register contains a data character. In FIFO mode, TEMT is set to 1 when the transmitter FIFO and the Transmit Shift register are both empty.
5	TDRQ	Transmit Data Request: TDRQ indicates that the UART is ready to accept a new character for transmission. In Non-FIFO mode, The TDRQ bit is set to logic 1 when a character is transferred from the Transmit Holding register into the Transmit Shift register. The bit is reset to logic 0 concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set to 1 when half of the characters in the FIFO have been loaded into the Shift register or the RESETTF bit in FCR has been set to 1. It is cleared when the FIFO has more than half data. If more than 64 characters are loaded into the FIFO, the excess characters are lost.



Register		LSR (Sheet 2 of 2)
Bits	Name	Description
4	BI	Break Interrupt: BI is set to a logic 1 when the received data input is held in the spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + parity bit + stop bits). The Break indicator is reset when the processor reads the Line Status Register. In FIFO mode, only one character (equal to 00H) is loaded into the FIFO regardless of the length of the break condition. BI shows the break condition for the character at the bottom of the FIFO, not the most recently received character.
3	FE	Framing Error: FE indicates that the received character did not have a valid stop bit. FE is set to a logic 1 when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). The FE indicator is reset when the processor reads the Line Status Register. The UART will resynchronize after a framing error. To do this, it assumes that the framing error was due to the next start bit, so it samples this "start" bit twice and then takes in the "data". In FIFO mode, FE shows a framing error for the character at the bottom of the FIFO, not for the most recently received character.
2	PE	Parity Error: PE indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set to logic 1 upon detection of a parity error and is reset to logic 0 when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the bottom of the FIFO, not the most recently received character.
1	OE	Overrun Error: In non-FIFO mode, OE indicates that the processor did not read data the receiver buffer register before the next character was received. The new character is lost. In FIFO mode, OE indicates that all 64 bytes of the FIFO are full and the most recently received byte has been discarded. The OE indicator is set to logic 1 upon detection of an overrun condition and reset when the processor reads the Line Status register.
0	DR	Data Ready: Bit 0 is set to logic 1 when a complete incoming character has been received and transferred into the receiver buffer register or the FIFO. In non-FIFO mode, DR is reset to 0 when the receive buffer is read. In FIFO mode, DR is reset to a logic 0 if the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in the FCR. 0 = No data has been received 1 = Data is available in RBR or the FIFO

14.5.11 Modem Status Register

This register provides the current state of the control lines from the modem or data set (or a peripheral device emulating a modem) to the processor. In addition to this current state information, four bits of the Modem Status register provide change information. The 3:0 bits are set to a logic 1 when a control input from the modem changes state. The bits are reset to a logic 0 when the processor writes ones to the bits of the Modem Status Register.

Note: When bits 0, 1, 2, or 3 are set to logic 1, a Modem Status Interrupt is generated, if bit 3 of the Interrupt Enable Register is set.



Register Name:		MSR																								
Hex Offset Address:		0xC800 X018								Reset Hex Value:		0x00000000														
Register Description:		Modem Status Register																								
Access: Read Only.																										
31																	8	7	6	5	4	3	2	1	0	
(Reserved)																			DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS



Register Name:		SPR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Hex Offset Address:		0xC800 X01C								Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
Register Description:		Scratch-Pad Register																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Access: Read/Write.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					

Register		SPR
Bits	Name	Description
31:8		(Reserved)
7	SCR	No effect on UART functionality

14.5.13 Infrared Selection Register

The Slow Infrared (SIR) Interface can be used in conjunction with a standard UART to support two-way, wireless communications using infrared radiation. It provides a transmit encoder and receive decoder to support a physical link that conforms to the Infrared Data Association Serial Infrared Specification.

In the IXP45X/IXP46X network processors, infrared mode is not supported. The reason for including this register in the address map and register description section is so software can ensure that this mode is never enabled.

Register Name:		ISR																									
Hex Offset Address:		0xC800 X020								Reset Hex Value:		0x00000000															
Register Description:		Infrared Selection Register																									
Access: Read/Write.																											
31																		8	7	6	5	4	3	2	1	0	
(Reserved)																				RXPL		TXPL		XMODE		(Reserved)	

Register		ISR (Sheet 1 of 2)
Bits	Name	Description
31:5		(Reserved)
4	RXPL	Receive Data Polarity: 0 = SIR decoder takes positive pulses as zeros 1 = SIR decoder takes negative pulses as zeros
3	TXPL	Transmit Data Polarity: 0 = SIR encoder generates a positive pulse for a data bit of zero 1 = SIR encoder generates a negative pulse for a data bit of zero



Register		ISR (Sheet 2 of 2)
Bits	Name	Description
2	XMODE	Transmit Pulse Width Select: When XMODE is set to 0, clocking of the IRDA transmit and receive logic is done by the UART clock, which must be operating in the 16X mode. When XMODE is set to a 1, the operation of the receive decoder does not change. The transmit encoder, however, generates pulses 1.6 μs wide (which is three clock periods at frequency 1.8432 MHz) — instead of the normal 3/16th of a bit time wide. The shorter infrared pulse — generated with XMODE set to 1 — reduces the power consumed by the LEDs. At 2,400 bps, the LED would normally be activated for 78 μs for each 0 bit transmitted. With XMODE set, the LED would be activated for only 1.6 μs. This would reduce the power consumed by the LED by a factor of almost 48. 0 = Transmit pulse width is 3/16th of a bit time wide 1 = Transmit pulse with is 1.6μs
1		(Reserved)
0		(Reserved)



15.0 GPIO Controller

15.1 Overview

The purpose of this document is to outline the functional requirements of the General-Purpose Input/Output (GPIO) pins for the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors.

The IXP45X/IXP46X network processors provide 16 general-purpose input/output pins for use in generating and capturing application specific input and output signals. Each pin can be programmed as either an input or output, and when GPIO0 through GPIO12 are programmed as an input, they can be used as an interrupt source. In addition, two of the pins can be programmed to provide a user programmable frequency source. During a reset all pins are configured as inputs and remain in this state until configured otherwise, with the exception of GPIO15, which by default will provide a clock output. Each GPIO pin is capable of driving external LEDs. There are eight distinct register functions used in the GPIO module. When used as an interrupt source, each pin can detect interrupts as active high, active low, rising edge, falling edge, or transitional.

Note: In this document, the NPE (Network Processing Engine) microprocessor core is also referred to as PSM or PSM2. The preferred term is NPE core.

Another two signals are going to the Tsync unit through the GPIO Controller, these two are snapshot signals for the Tsync unit. They are wired through the GPIO, and goes to sync cells in the tsync unit. The Auxiliary slave snapshot signal is connected to GPIO_IN[8] and the Auxiliary master snapshot signal is connected to GPIO_IN[7].

15.2 Feature List

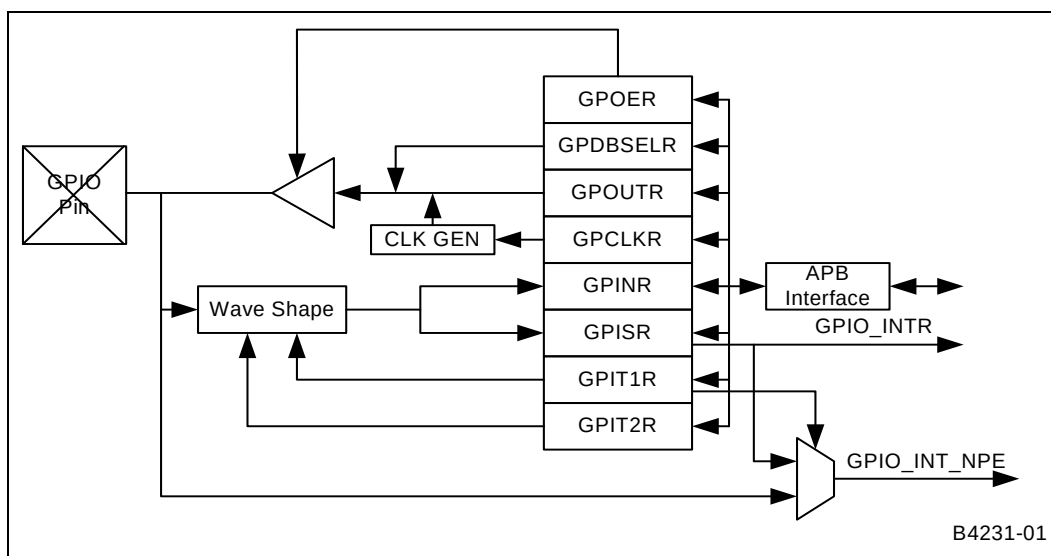
- Sixteen GPIO pins.
- Thirteen pins capable of being an interrupt source.
- Each pin can be programmed to be an input or output.
- Two pins can be programmed as a programmable clock source.
- Eight different functional registers.
- AMBA APB interface.
- NPE core and IEEE1588 debug functionality.
- Two snapshot trigger inputs.
- GPIO_IN[7:0] can be individually routed directly to the NPEs.

15.3 Block Diagram

A block diagram of the General-Purpose I/O block is shown in [Figure 185](#).



Figure 185. GPIO Block Diagram



15.4 Theory of Operation

The GPIO pin is controlled through six registers and two more registers provide status. Each register can be read through the APB interface and all registers except GPINR can be written through the APB interface.

The GPIO on the IXP45X/IXP46X network processors can be configured to be used as general-purpose inputs or general-purpose outputs. Three 16-bit registers are used in order to configure, initialize, and use the general-purpose I/O. These registers are:

- General-Purpose Data Output Register (GPOUTR)
- General-Purpose Output Enable Register (GPOER)
- General-Purpose Input Status Register (GPINR)

The General-Purpose Output Enable Register is used to configure the GPIO pins as an input or an output. There is a one-for-one relationship between the register bit mapping and the GPIO. For example, bit 0 of the general-purpose Output Enable Register corresponds to GPIO 0 and bit 1 of the General-Purpose Output Enable Register corresponds to GPIO 1.

When a bit of the General-Purpose Enable Register contains logic 0, the corresponding GPIO will be configured as an output. A logic 1 in the same bit of the General-Purpose Enable Register will cause the corresponding GPIO to be configured as an input.

For example, the General-Purpose Output Enable Register contains a hexadecimal value of 0x00000500. GPIO 8 and GPIO 10 will be configured as inputs and all other GPIO will be configured as outputs. The GPIO that are configured as outputs — by the values contained in the General-Purpose Enable Register — will be driven by the values contained in the General-Purpose Data Output Register.

The General-Purpose Data Output Register is a 16-bit register with a one-for-one correspondence between the 16 bits of the General-Purpose Data Output Register and the 16-bit GPIO. When logic 1 is written to a bit in the General-Purpose Data Output Register — and the corresponding bit in the General-Purpose Enable Register is set to logic 0 — logic 1 will be replicated to the corresponding GPIO. When logic 0 is written to



the same bit in the General-Purpose Data Output Register — and the corresponding bit in the General-Purpose Enable Register is still set to a logic 0 — logic 0 will be replicated to the corresponding GPIO.

For example, the General-Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General-Purpose Data Output Register is set to hexadecimal 0x00000401. GPIO8 and GPIO 10 will be configured as outputs, GPIO8 will drive logic 0, and GPIO10 will drive logic 1.

Notice that bit 0 of the General-Purpose Data Output Register is set to logic 1. However, GPIO0 is configured as an input so the logic 1 in the General-Purpose Data Output Register is not replicated to GPIO0.

Reading of the current status of the GPIO can be obtained by reading the General-Purpose Input Status Register. The General-Purpose Input Status Register is a 16-bit register with a one-for-one correspondence between the 16 bits of the General-Purpose Input Status Register and the 16-bit GPIO.

When reading the General-Purpose Input Status Register, the current logic value of the GPIO will be reflected in the corresponding bits of the General-Purpose Input Status Register. The values will be replicated regardless of the configuration of the GPIO as defined by the General-Purpose Enable Register.

If a GPIO is configured as an output, the value driven to the output will be read when reading the corresponding bits of the General-Purpose Input Status Register. For example, the General-Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General-Purpose Data Output Register is set to hexadecimal 0x00000401 and the GPIO pins have the following signals being supplied as inputs or driven as outputs, hexadecimal 0xAE37 with GPIO[15:12] = A and GPIO[3:0] = 7. GPIO8 and GPIO10 are configured as outputs as defined by the General-Purpose Enable Register. All other GPIO pins are configured as inputs. When the General-Purpose Input Status Register is read the value of hexadecimal 0x0000AE37 will be returned.

15.4.1 Input Meta-Stability Protection, Edge Detect Logic, Pulse Discrimination

The GPIO input pin will be sampled through a synchronizer cell to provide meta-stability protection.

For the interrupt detection logic the pulse width of the synchronized GPIO input is monitored to ensure that it is greater than four pclk cycles. This discriminates against small pulses and ensures that glitches are not detected. A counter based approach is used to monitor the pulse width. A valid condition, which depends on the interrupt detect type programmed in the GPIT registers, is used to enable the counter. Once the count has reached four pclk cycles the appropriate bit is set in the GPISR register. Each programmed input pin of GPIO0 through GPIO 12 can detect active high, active low, rising, falling or transitional (rising/falling) interrupts.

15.4.2 Clock Generation

The GPIO block provides two programmable clock outputs. The clocks are generated by counters based on the APB clock, pclk. There are two register settings which control the operation of the clock generator, one to define the terminal count (TC) of the counter, the other to define the duty cycle (DC). The counters are 4-bit up counters (i.e., counting from zero to the TC), free running at the pclk rate. Therefore, the TC defines the clock period. The Duty Cycle (DC) represents the number of counts for which output clock is /ow. There is one special condition defined; if TC = F and DC = F, then the output of the clock generator block will be pclk/2. If the duty cycle (DC) is programmed to be greater than or equal to the Terminal Count (TC), the generator outputs a '1'.



The clock generation logic is reset using an early reset, *early_reset_n* – this reset is de-asserted prior to system reset being de-asserted. This ensures that GPIO15 provides a clock out while system reset is still asserted.

15.4.3 APB Interface

The GPIO block interfaces to the Advanced Peripheral Bus, APB. The AHB/APB Bridge provides the interface timing/signals required. Refer to the AMBA Rev 2.0 specification for a detailed description on the Advanced Micro controller Bus Architecture. The GPIO block responds to 8, 16, and 32 bit reads and writes.

15.5 Detailed Register Descriptions

Table 252. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 253. Register Summary

Apb_paddr [31:0]	Register Name	Description	Reset Value	Attribute
"0xC8004000"	GPOUTR	GPIO pin data output register	"0x00000000"	RW
"0xC8004004"	GPOER	GPIO pin out enable register	"0x00007FFF"	RW
"0xC8004008"	GPINR	GPIO pin status register	"0x00000000"	RO
"0xC800400C"	GPISR	GPIO interrupt status register	"0x00000000"	RW1C
"0xC8004010"	GPIT1R	GPIO interrupt type register, inputs 7:0	"0x00000000"	RW
"0xC8004014"	GPIT2R	GPIO pin interrupt type register, inputs 15:8	"0x00000000"	RW
"0xC8004018"	GPCLKR	GPIO Clock Control Register	"0x01100000"	RW

The internal registers are accessed through the APB interface. [Table 253](#) presents the registers.

15.5.1 GPIO Output Register

Each pin's output data is controlled by programming this register. Each of the 16 bits in the register represents the data to be put on the output through a tri-state buffer, depending upon the status of the GPOER. The register is read and written to through the APB interface on the rising edge of apb_pclk.



Register Name:		GPOUTR																														
Physical Address:		0xC8004000				Reset Hex Value:				0x00000000																						
Register Description:		I/O Output register. Controls output value of GPIO pins, depending on tri-state control from GPOER.																														
Access: Read/Write																																
31																16	15															0
(Reserved)														DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0			

Register		GPOUTR			
Bits	Name	Description	Reset Value	Access	
[31: 16]	(Reserved)	Reads back 0	0x0	RO	
15	DO15	1 = Output a 1 on output pin, depends on GPCLKR24, GPOER15 0 = Output a 0 on output pin, depends on GPCLKR24, GPOER15	0	RW	
14	DO14	1 = Output a 1 on output pin, depends on GPCLKR8, GPOER14 0 = Output a 0 on output pin, depends on GPCLKR8, GPOER14	0	RW	
13:9	DO13:DO9	1 = Output a 1 on output pin, depends on GPDBSEL2, GPOER[13:9] 0 = Output a 0 on output pin, depends on GPDBSEL2, GPOER[13:9]	0	RW	
8	DO8	1 = Output a 1 on output pin, depends on testmode_data, GPOER[8] 0 = Output a 0 on output pin, depends on testmode_data, GPOER[8]	0	RW	
7:0	DO7:DO0	1 = Output a 1 on output pin GPOER[7:0] 0 = Output a 0 on output pin GPOER[7:0]	0	RW	

15.5.2 GPIO Output Enable Register

Each pin's output tri-state buffer is controlled by programming this register. Each of the 16 bits in the register represents the control to the tri-state buffer of the output pin. The register is read and written to through the APB interface on the rising edge of apb_pclk.

Register Name:		GPOER																														
Physical Address:		0xC8004004				Reset Hex Value:				0x00007FFF																						
Register Description:		I/O Output Enable register. Turns on output driver.																														
Access: Read/Write																																
31																16	15															0
(Reserved)												OE15	OE14	OE13	OE12	OE11	OE10	OE9	OE8	OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0					



Register		GPOER		
Bits	Name	Description	Reset Value	Access
31: 16	(Reserved)	Reads back 0	0x0	RO
15	OE15	1 = Output pin is tri-stated or input 0 = Output pin is driven	0 (as clock is driven out on this pin during reset)	RW
14:0	OE14:OE0	1 = Output pin is tri-stated or input 0 = Output pin is driven	1	RW

15.5.3 GPIO Input Register

This is a read only register. This register contains the level of the I/O pin, either a '1' or a '0'.

Register Name:		GPINR																										
Physical Address:		0xC8004008						Reset Hex Value:				0x00000000																
Register Description:		This register is used to monitor input pins.																										
Access: Read																												
3 1													1 6	1 5							8	7						0
(Reserved)												IN_LEV																

Register		GPINR		
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
15:0	IN_LEV	Level of general-purpose inputs 15-0 1 = 1 on GPIO 0 = 0 on GPIO	0x0000	RO

15.5.4 GPIO Interrupt Status Register

This register is used to store status of a GP input interpreted as an interrupt. GP input interrupts can be configured as active high, active low, rising edge, falling edge, or transitional depending upon the configuration of the GPIT[1:0]R register. A '1' read from this register indicates a pending interrupt. Writing a '1' back to this register will clear the interrupt provided the interrupting condition no longer exists. The interrupts are all masked in the Interrupt Controller block.

Note: GPIO0 through GPIO12 can be used as an interrupt (see [Table 261, “Intel XScale® Processor Interrupt Mapping” on page 804](#))



Register Name:		GPISR																	
Physical Address:		0xC800400C								Reset Hex Value:		0x00000000							
Register Description:		This register is used to store status of interrupts received on GP input pins																	
Access: Read/Write																			
3																	1	1	
1																	6	5	
(Reserved)										INT_STAT									

Register		GPISR		
Bits	Name	Description	Reset Value	Access
31:1 6	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
15:1 3	(Reserved)	Not used.	0x0	RO
12:0	INT_STAT	1 = Interrupt pending. 0 = No interrupt pending.	0x0000	RW1C

15.5.5 GPIO Interrupt Type Register 1

This register describes how to interpret GPIO [7:0] as interrupts, either level or edge, along with high, low, rising, falling, transitional. Three bits describe each GPIO pin, as described in the following table. The top [31:24] bits are used to control muxing between raw data from GPIO_IN[7:0] or GPISR[7:0] to the NPEs.

Register Name:								GPIT1R																							
Physical Address:								0xC8004010				Reset Hex Value:				0x00000000															
Register Description:								This register is used to control interrupt type for GPIO 7:0																							
Access: Read/Write																															
3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3							1 6	1 5							8	7							0
gpio_npe_7	gpio_npe_6	gpio_npe_5	gpio_npe_4	gpio_npe_3	gpio_npe_2	gpio_npe_1	gpio_npe_0	GPIO7			GPIO6			GPIO5			GPIO4			GPIO3			GPIO2		GPIO1			GPIO0			

Register		GPIT1R (Sheet 1 of 2)		
Bits	Name	Description	Reset Value	Access
31	gpio_npe_7	When '1', a synchronized gpio_in[7] is muxed to gpio_int_npe[7], when '0', gpisr[7] is muxed to gpio_int_npe[7]	0	RW
30	gpio_npe_6	as per gpio_npe_7	0	RW
29	gpio_npe_5	as per gpio_npe_7	0	RW
28	gpio_npe_4	as per gpio_npe_7	0	RW



Register		GPIT1R (Sheet 2 of 2)		
Bits	Name	Description	Reset Value	Access
27	gpio_npe_3	as per gpio_npe_7	0	RW
26	gpio_npe_2	as per gpio_npe_7	0	RW
25	gpio_npe_1	as per gpio_npe_7	0	RW
24	gpio_npe_0	as per gpio_npe_7	0	RW
23:2 1	GPIO7	000 – Active High 001 – Active Low 010 – Rising Edge 011 – Falling Edge 1xx – Transitional	000 Active High	RW
20:1 8	GPIO6	As per GPIO7	000 Active High	RW
17:1 5	GPIO5	As per GPIO7	000 Active High	RW
14:1 2	GPIO4	As per GPIO7	000 Active High	RW
11:9	GPIO3	As per GPIO7	000 Active High	RW
8:6	GPIO2	As per GPIO7	000 Active High	RW
5:3	GPIO1	As per GPIO7	000 Active High	RW
2:0	GPIO0	As per GPIO7	000 Active High	RW

15.5.6 GPIO Interrupt Type Register 2

This register describes how to interpret GPIO[15:8] as interrupts, either level or edge, along with high, low, rising, falling, transitional three bits describe each GPIO pin, as described in the following table.

Register Name:		GPIT2R																										
Physical Address:		0xC8004014								Reset Hex Value:				0x00000000														
Register Description:		This register is used to control interrupt type for GPIO 15:8																										
Access: Read/Write																												
3 1													1 6	1 5							8	7						0
(Reserved)								GPIO15		GPIO14		GPIO13		GPIO12		GPIO11		GPIO10		GPIO9		GPIO8						



Register		GPIT2R		
Bits	Name	Description	Reset Value	Access
31:2 4	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
23:2 1	GPIO15	Not used	0x0	RW
20:1 8	GPIO14	Not used	0x0	RW
17:1 5	GPIO13	Not used	0x0	RW
14:1 2	GPIO12	000 – Active High 001 – Active Low 010 – Rising Edge 011 – Falling Edge 1xx – Transitional	000 Active High	RW
11:9	GPIO11	As per GPIO12	000 Active High	RW
8:6	GPIO10	As per GPIO12	000 Active High	RW
5:3	GPIO9	As per GPIO12	000 Active High	RW
2:0	GPIO8	As per GPIO12	000 Active High	RW

15.5.7 GPIO Clock Register

This register controls the use of GPIO15 and GPIO14 as clock outputs. GPOER defines the output enable for the driver; this register controls both the clock dividers and a MUX between the clock data and the data defined in GPOUTR. This register defines two four bit counts for a clock divider running from the pclk clock, one defines the period of the clock and the other defines the duty cycle. After early reset has been deasserted, GPIO 15 will be defined as a pclk/2 output clock (33 MHz assuming 66 MHz pclk), with the expectation that this might be used to control the expansion bus peripherals. The expansion bus requires an external clock to operate; it could be provided from GPIO15 or an external frequency source. As defined in GPOER, GPIO14 is an input at reset. There are two special cases for these counters. First, if the TC is defined as '0' then the counter is disabled and the clock output will be high. Second, if both TC and DC are 'F', then the output clock will be pclk/2. If DC is = TC, then the clock output will be high.

Register Name:		GPCLKR																											
Physical Address:		0xC8004018				Reset Hex Value:				0x01100000																			
Register Description:		This register controls the use of GPIO 15 and GPIO14 as clock sources																											
Access: Read/Write																													
3 1													1 6	1 5							8	7							0
(Reserved)						Mx15	CLK1TC				CLK1DC						Mx14	CLK0TC				CLK0DC							



Register		GPCLKR		
Bits	Name	Description	Reset Value	Access
31:2 5	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
24	MUX15	0 – Control from GPOUTR Register 1 – Clock output	1	RW
23:2 0	CLK1TC	Terminal count for a 4 bit up counter @ PCLK. An 'F' in this field and the CLK1DC field is a special case to provide PCLK/2.	0x1	RW
19:1 6	CLK1DC	Represents the number of counts for which clock output should be low	0x0	RW
15:9	-	Not used. Ignored on writes and driven logic '0' on reads.		RW
8	MUX14	0 – Control from GPOUTR Register 1 – Clock Output	0	RW
7:4	CLK0TC	Terminal count for a 4 bit up counter @ PCLK. An 'F' in this field and the CLK0DC field is a special case to provide PCLK/2.	0x0	RW
3:0	CLK0DC	Represents the number of counts for which clock output should be low	0x0	RW

§ §





16.0 Performance Monitoring Unit (PMU)

16.1 Overview

This chapter describes the components which comprise the Performance Monitoring Unit (PMU). These features aid in measuring and monitoring various system parameters that contribute to the overall performance of the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. Operation modes, setup mechanisms, registers, and interrupts are also described in this chapter.

The Performance Monitoring (PMON) facility provided by the PMU comprises:

- Eight Programmable Event Counters (PECx)
- Previous Master/Slave Register
- Event-selection mux

The programmable event counters are 27 bits wide. Each counter may be programmed to observe one event from a defined set of events. An event consists of a set of parameters which define a condition. The monitored events are selected by programming the Event Select Registers (ESRx).

The PMU is inherently a model specific function. This definition attempts to make the design and software model of the PMU more regular. However, the definition of events between instantiations of the PMU on various members of the family of network processors can be expected to change. However this specification makes access to these events highly regular and allows a simple table driven approach to associating the programming to event specification mapping.

There is lots of empty space in the event address map which will enable future expansion.

16.2 Feature List

The major features of the PMU are:

- Eight Programmable Event Count Registers
- Duration and Event detection logic
- Monitoring mechanisms
- Event selection logic
- Previous accessing/accessed Master/ Slave Register

16.3 Functional Description

The PMU comprises:

- Eight Programmable Event Counters (PEC).
- Event Selection logic



- Overflow detection and interrupt
- Previous Master and Slave information

The event counters are used to monitor events across different interfaces of the processor. The counters do nothing special to the events, only count the number of cycles that the event is “TRUE”. What makes an event a specific kind of performance event is the logic which decodes operations from the monitored buses and creates event streams to present to the event selection mux.

Unlike previous versions of the PMU, there are no “modes” or really any meaningful difference between duration and occurrence events. There are only events and each event has its description which defines what parameter is being measured. Some events are defined as occurrence events and some are defined as a duration event, but the implementation or programming of the PMU does not distinguish between any of the cases. From the software and implementation perspective, all events are uniform once they are presented to the selection mux. An occurrence event occurs only once per occurrence and back-to-back events represent two occurrences. A duration event is active the entire time the event is “TRUE.” Back-to-back events may either represent two disconnected events, back-to-back, or may represent consecutive cycles of the same event. Occurrence events have the semantics of a count whereas duration events have the semantics of time.

16.3.1 Programmable Event Counters

There are eight general-purpose, 27-bit wide Programmable Event Counters (PECx). The reason for the size of the counter is to provide a count period of approximately one second at 133 MHz. Each counter may be programmed to increment on detection of a rising edge or continuously with a high level on the selected input signal. If any counter generates an overflow, a flag is set and an interrupt sent to the interrupt controller. The flag is cleared on a write to the PMU status register. The counters are reset whenever a new mode is selected, (i.e., North, South or DRAM). Mode HALT (i.e. PMR set to disabled) causes all counters to halt allowing the same time snap-shot to be read across all counters.

The event that is being monitored by a counter is dependent on the ESR contents. There is an eight bit field per counter which selects the event being presented to each counter. In this way up to eight events may be selected for monitoring. There is some redundancy between counter events to allow as many useful combinations to be monitored as possible (see [Table 260](#) for combinations).

The programmable event counters provide real-time monitoring capability. They are memory mapped and may be read directly. All counters are reset and started whenever a value is written to the ESR and the mode is not HALT.

16.3.2 Occurrence Events

An occurrence event causes the counter to increase by one each time the event occurs. [Table 254](#) presents the various occurrence events that are monitored on the IXP45X/ IXP46X network processors. Note that the PMU monitors two AHB buses, the north and south. The logic performing the monitoring does not distinguish between the two buses, and the only effect that multiple buses has is to increase the number of events.



Table 254. Occurrence Events

Observed Interface	Monitored Event	Description
AHB (Either North or South)	Number of grants to the particular AHB device. Name: "xAHB[N] Grant Occur"	Monitors the number of times a master is granted the bus. It increments the counter when the master is the bus initiator. The counter is incremented once for every new transaction. For multi-cycle transactions, the counter increments once on the first cycle. Condition: HMASTER = MstrX
	Number of transactions initiated by the particular AHB device. Name: "xAHB[N] Xfer Occur"	Increments the counter every time that a master initiates a transaction on the bus. This event primitive may be used in conjunction with the next primitive (retry), to calculate the effectiveness of transactions initiated on the AHB Bus by the master (i.e., % transactions initiated by the master that are not retried.). Condition: HMASTER = MstrX & HTRANS = NonSeq
	Number of retries Signaled by the particular AHB device to the initiators. Note that not all AHB devices may initiate. Name: "xAHB[N] Retry Occur"	Increments the counter every time that a transaction initiated by the master is responded to with a Retry by the target. Condition: HMASTER = MstrX & HRESP = Retry
	Number of split transfers claimed by the particular AHB device to the initiators. Note that not all AHB devices may initiate. Name: "xAHB[N] Split Occur"	Counts the number of times that SlaveX claims a Split Transaction. Condition: Hsel_SlaveX = 1 & HRESP = Split
MPI	Number of grants to the particular MPI port. Name: "MPI[N] Grant Occur"	Counts the number of times that PortX is granted an MPI transaction Condition: mpi_ackX
	Number of reads to the particular MPI port. Name: "MPI[N] Rd Occur"	Counts the number of times that PortX issues a read cycle Condition: mpi_ackX & !mpi_wr_rden_n
	Number of writes to the particular MPI port. Name: MPI[N] Wr Occur"	Counts the number of times that PortX issues a write cycle Condition: mpi_ackX & mpi_wr_rden_n
MEM	Number of occurrences of Event[N] from the MCU. Name: "MCU[N]"	This event bus encodes up to 16 occurrence events where only one event may be active in any given system clock cycle. This is used to count page hits and page missed for the 8 DRAM pages.
	Latency Events Name: "Universal Latency N"	Using the universal latency event bus things like the read latency of multiple outstanding reads may be measured.

16.3.3 Duration Events

For a duration event, the counter counts the number of clocks during which a particular condition or set of conditions is true. Duration measurements comprise:

- **Bus Acquisition Latency** — Represents the elapsed time between a bus master issuing a request for the bus to when it is granted the bus.



- **Split Transfer Latency** — Represents the elapsed time between the Requester receiving a Split Response and the Split claimer transferring the first DWORD of Data Read.
- **Bus Ownership Time** — Represents the elapsed time during which an initiator is in control of the bus.
- **Initiator Bus Data Transfer Time** — Represents the elapsed time during which an initiator transfers data.
- **Split Data Transfer Time** — Represents the elapsed time during which a Split claimer transfers data (Data Read) to a requester during a split transfer transaction. Please note that this metric is associated with the requester and **not** the split claimer.

Table 255 lists the various duration events that are monitored. The *Name* corresponds to the entry in Table 260.

Table 255. Duration Events (Sheet 1 of 2)

Observed Interface	Monitored Event	Description
AHB, Any Device	Number of clocks the AHB bus is doing Data Writes. Name: "AHBx Write"	Increments the counter on every AHB Bus Write data cycle. This enables calculation of data utilization of the Write data bus. Condition: HWrite = 1 & HReady = 1 & HTRANS = (seq OR NonSeq))
	Number of clocks the AHB bus is doing Data Reads. Name: "AHBx Read"	Increments the counter on every AHB Bus Read data cycle. This enables calculation of data utilization of the Read data bus. Condition: HWrite = 0 & HReady = 1 & HTRANS = seq or NonSeq
	Number of clocks the AHB bus is Idle. Name: "AHBx Idle"	Increments the counter every AHB Bus idle cycle. An idle cycle occurs when there is no activity on the bus due to data being transferred and the bus is not in an overhead cycle. An overhead cycle is a cycle when an initiator owns the bus, however the initiator is unable to send data or the target is unable to receive data - hence no data is transferred. Condition: All Grant signals de-asserted
Specific AHB Device	Bus ownership for the AHB Masters. Name: "xAHB[N] Grant Duration"	Counts the number of clocks spent by a master granted the AHB bus.
	Bus ownership metrics for the AHB Masters. Name: "xAHB[N] Own Duration"	Counts the duration for which the master is the initiator on the AHB bus. The counter increments on every clock cycle during which the master is the bus initiator.
	Initiator data transfer time for Writes on the AHB Masters. Name: "xAHB[N] WriteDuration"	Increments counter every AHB bus write data cycle where the master is the initiator.
	Initiator data transfer time for Reads on the AHB Masters. Name: "xAHB[N] Read Duration"	Increments counter every AHB bus read data cycle where the master is the initiator.



Table 255. Duration Events (Sheet 2 of 2)

Observed Interface	Monitored Event	Description
MPI	Number of clocks the MPI port is doing Data Writes. Name: "MPI[N] Wr Duration"	Increments counter for every cycle MPI port is transferring write data.
	Number of clocks the MPI port is doing Data Reads. Name: "MPI[N] rDDuration"	Increments counter for every cycle MPI port is transferring write data.
	Number of clocks the MPI port is idle. Name: "MPI[N] IdleDuration"	Increments counter for every cycle MPI port is not requesting data nor transferring data.
	Bus acquisition latency for the MPI port. Name: "MPI[N] Latency Duration"	Increments counter for every cycle MPI port is requesting a transfer but not acknowledged.

16.3.4 Performance Monitoring

Performance Monitoring consists of a collection of event primitives, which may then be used for statistical calculations.

Different events on the AHBs are monitored by programming the ESR. Monitoring these events provide information about the AHB and the AHB initiators.

16.3.4.1 Halt: Performance Monitoring Disabled

Halting one or more counters disables performance monitoring on the halted counter. In this way any combination or all counters are halted at the same time. Counters are reset whenever a new mode (not Halt) is selected. Halt/enable is selected on a per counter basis by the enable bits in the PMR. Halt/enable conditions are maintained in a separate register from the mux programming because not all of the counter mux registers can be updated in the same cycle.

The normal expected mode of operation is to halt all counters, program in the counter event mux selects, and then enable the desired counters.

16.3.4.2 Cycle Count

Selecting mux input 0xFF (event select == 256) simply selects the event 'TRUE' which has the effect of putting the counter into continuous count mode.

16.3.4.3 MCU: DRAM Transactions

Selecting MCU events enables performance monitoring of the DRAM. All counters are clocked at the AHB frequency. These transactions are measured using event signals provided from the DRAM controller. See [Chapter 11.0, "Memory Controller"](#) for information on the semantics of these event signals.

By setting up the IXP45X/IXP46X network processors system-level PMU registers, the following MCU performance parameters can be monitored:

- Event type 0 : PMU registers ESR0, ESR1 programmed to 0x00000000
PCEC [0-7] contains the page **miss** counts for each of the 8 possible open pages of DDR SDRAM.
- Event type 1: PMU registers ESR0, ESR1 programmed to 0x01010101
PCEC [0-7] contains the page **hit** counts for each of the 8 possible open pages of DDR SDRAM.



- Event type 39: PMU registers ESR0, ESR1 programmed to 0x27272727
PCEC [0-7] contains the core memory bus read latency (from Intel XScale processor to DDR request to valid data back for each of the eight outstanding read requests possible from the Intel XScale processor to memory).
- Event type 40: PMU register ESR0 programmed to 0x00002828
PEC0 (AHB North) and PEC1 (AHB South) contains the AHB bus Read latency (from request to valid data back for each of the AHB bus ports).

Note: The MCU is PMU event 3 on AHB North and event 5 on AHB South.

16.3.4.4 Events

When not halted, a counter will count the cycles an event is “TRUE.” The event descriptions are listed above. To alleviate the wiring of redundant events into a very large event mux, not all events are connected to every counter. Instead, the allocation of events is intended to be a compromise between flexibility and simplicity. The basic philosophy is that you should be able to measure all attributes about a given device, all of the same attributes about a number of devices, or a subset of the two. An attempt is made to maximize the flexibility but any conceivable combination of performance metrics may not fit into 8 counter's worth of information or the chosen eight may not be all accessible at the same time. To acquire any given group of metrics, it may be necessary to repeat two passes of the same operations. There are some natural inaccuracies that will result as a consequence of multiple passes unless extreme care is taken to make the two passes identical.

16.4 Previous Master and Slave

The PMU provides a register that indicates the last masters and the slave they accessed on both the North and South AHB busses. The master value stored is determined from the HMASTER signal, which is simply the grants to the master devices. Whenever this signal changes and there is a bus transaction the register bits will update. The slave value stored is determined from the HSELx signal. Whenever the slave selected changes and there is a bus transaction these register bits will update.

The PMSR will also lock the current master/slave in the case of an AHB error signal. There are three reasons for an AHB error signal, an unsupported bus operation to a slave, an uncorrectable error by a slave, and an out-of-range address. In the first two cases, the HSEL field will contain the device which was being addressed, and in the third case the HSEL field will contain 0xF. The first occurrence of an error signal on a bus will cause the master and slave fields to be updated to the current value, the “stuck” bit appropriate for the north and south fields will be set, and further updates blocked until the “stuck” bit has been reset (by writing a ‘1’ to the bit position). Multiple error conditions are not supported, and any subsequent errors after the first are ignored.

16.5 Miscellaneous

16.5.1 Interrupts

Each of the counters has an associated overflow flag. These flags are readable from the PMU Status Register (PSR). The pmu_overflow_int interrupt is the logic OR of all the overflow flags. The flags and interrupt are cleared when the register is written with ones in the overflow positions. Setting an overflow condition from the counter takes precedence over resetting the interrupt. If the counter overflows on the exact same cycle the status is being reset, the overflow will be set.



16.5.2 Reset Conditions

The ESR defaults to Mode Halt upon reset: performance monitoring is disabled and all counters are disabled in this mode. The Programmable Event Counters (PECRx) values are cleared upon reset.

16.6 Detailed Register Descriptions

Table 256. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

The performance monitoring facility on the IXP45X/IXP46X network processors consists of 13 memory-mapped registers for controlling operation and monitoring various events. Each register appears to be 32-bits wide to the APB bus. Each of these registers is accessed as a memory-mapped 32-bit register with a unique memory address. Access is accomplished through regular memory-format instructions from the Bus Interface Unit.

Table 257 presents the registers and their offset address.

Table 257. PMU Register Table

Register Name	Reset Hex Value	Hex Offset Address
ESR0	0x00000000	0x00000000
ESR1	0x00000000	0x00000004
(Reserved)		0x00000008
(Reserved)		0x0000000C
PSR	0x00000000	0x00000010
PMR	0x00000000	0x00000014
PMSR	0x00000000	0x00000018
(Reserved)		0x0000001C
PEC0	0x00000000	0x00000020
PEC1	0x00000000	0x00000024
PEC2	0x00000000	0x00000028
PEC3	0x00000000	0x0000002C
PEC4	0x00000000	0x00000030
PEC5	0x00000000	0x00000034
PEC6	0x00000000	0x00000038
PEC7	0x00000000	0x0000003C



16.6.1 Event Select Registers

ESR0 and ESR1

The ESR controls the specific item being monitored. Each PECx field is programmed according to [Table 260, “Event Mux Programming” on page 799](#).

To change the monitored event, it is necessary to write the entire ESR. The Programmable Event Counters are reset and started when a new value is written to the ESR. However, notice that since there are two registers and one cannot write both of them in the same cycle, it is preferable to halt the counters via the MODE register (i.e., PMR), update the mux selects via these registers (which will reset the counters), and then enable the desired counters with the PMR.

Register Name:		ESR0																										
Physical Address:		0xC800 2000				Reset Hex Value:				0x00000000																		
Register Description:		Event Mux Select Register, counters 3-0																										
Access: Read/Write																												
31														16	15					8	7							0
PEC3 ctrl						PEC2 ctrl						PEC1 ctrl						PEC0 ctrl										

Register		ESR			
Bits	Name	Description	Reset Value	Access	
31:2 4	PEC3 ctrl	Selects Enable conditions for counter PEC3.	0xFF	RW	
23:1 6	PEC2 ctrl	Selects Enable conditions for counter PEC2.	0xFF	RW	
15:8	PEC1 ctrl	Selects Enable conditions for counter PEC1.	0xFF	RW	
7:0	PEC0 ctrl	Selects Enable conditions for counter PEC0.	0xFF	RW	

Register Name:		ESR1																											
Physical Address:		0xC800 2004				Reset Hex Value:				0x00000000																			
Register Description:		Event Mux Select Register, counters 7-4																											
Access: Read/Write																													
31														16	15						8	7							0
PEC7 ctrl						PEC6 ctrl						PEC5 ctrl						PEC4 ctrl											

Register		ESR (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	Access	
31:2 4	PEC7 ctrl	Selects Enable conditions for counter PEC7.	0xFF	RW	

Register		ESR (Sheet 2 of 2)		
Bits	Name	Description	Reset Value	Access
23:16	PEC6 ctrl	Selects Enable conditions for counter PEC6.	0xFF	RW
15:8	PEC5 ctrl	Selects Enable conditions for counter PEC5.	0xFF	RW
7:0	PEC4 ctrl	Selects Enable conditions for counter PEC4.	0xFF	RW

16.6.2 PMU Status Register

PSR

The PSR allows access to the over flow flags from the PEC counters. These flags remain set until cleared by writing a '1' to the bit. Setting an overflow condition from the counter takes precedence over resetting the interrupt. If the counter overflows on the exact same cycle the status is being reset, the overflow will be set. If any of these status bits are set, the corresponding counter has overflowed and an interrupt will continue to be generated until all bits are clear.

Register Name:		PSR																												
Physical Address:		0xC800 2010				Reset Hex Value:				0x00000000																				
Register Description:		Counter Overflow Status Register																												
Access: Read, Clear on write																														
31													16	15								8	7							0
(Reserved)																OFL7	OFL6	OFL5	OFL4	OFL3	OFL2	OFL1	OFL0							

Register		PSR		
Bits	Name	Description	Reset Value	Access
31:8	Reserved	Always zero.		
7	OFL7	1 = PEC7 has overflowed	0	RW1C
6	OFL6	1 = PEC6 has overflowed	0	RW1C
5	OFL5	1 = PEC5 has overflowed	0	RW1C
4	OFL4	1 = PEC4 has overflowed	0	RW1C
3	OFL3	1 = PEC3 has overflowed	0	RW1C
2	OFL2	1 = PEC2 has overflowed	0	RW1C
1	OFL1	1 = PEC1 has overflowed	0	RW1C
0	OFL0	1 = PEC0 has overflowed	0	RW1C

16.6.3 PMU Mode Register

PMR

The PMR controls which counters are enabled or halted. If the bit corresponding to a particular PEC is false, the counter is not enabled to count and if the bit is true, the PEC is enabled to count.



Register Name:		PMR																																											
Physical Address:		0xC800 2014								Reset Hex Value:				0x00000000																															
Register Description:		Counter Enable Mode Register																																											
Access: Read, Clear on write																																													
31																								16	15										8	7									0
(Reserved)																									Enable7		Enable6		Enable5		Enable4		Enable3		Enable2		Enable1		Enable0						

Register		PSR			
Bits	Name	Description	Reset Value	Access	
31:8	(Reserved)	Always zero.			
7	Enable7	1 = PEC7 is Enabled	0	RW	
6	Enable6	1 = PEC6 is Enabled	0	RW	
5	Enable5	1 = PEC5 is Enabled	0	RW	
4	Enable4	1 = PEC4 is Enabled	0	RW	
3	Enable3	1 = PEC3 is Enabled	0	RW	
2	Enable2	1 = PEC2 is Enabled	0	RW	
1	Enable1	1 = PEC1 is Enabled	0	RW	
0	Enable0	1 = PEC0 is Enabled	0	RW	

16.6.4 Programmable Event Counters

PECx

There are seven programmable event counters (PEC0 – PEC7) that are available through the memory map. These counters are 27-bit wide and are read only.

The value in any register is incremented based on the current programmed ESR value and the descriptions shown in the following tables in this subsection. When a new event to monitor is chosen by writing a value to the ESR, these registers are reset to zero. The counters are enabled via the PMR.



Register Name:		PECx																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
Physical Address:	0xC800 2020				Reset Hex Value:				0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
	0xC800 2024																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 2028																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 202C																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 2030																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 2034																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 2038																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
	0xC800 203C																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
Register Description:		Event Counter																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
Access: Read																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
31															16	15																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											

Register		PECx															
Bits	Name	Description														Reset Value	Access
31:2 7	(Reserved)	Always zero.															
26:0	PECx	This is a 27-bit, read-only counter register.														0x00000000	R

16.6.5 Previous Master / Slave Register

PMSR

This register encodes the device which previously accessed or was being accessed on the monitored buses.

This register is reset, but the reset value is quickly overwritten with the bus activity, so the reset value has little practical meaning. This value reflects the last recorded activity. To build up a histogram of master/slave pairs this register would be read periodically. However there is no way to know that an unchanging value means that there was no bus activity.

Register Name:		PMSR																														
Physical Address:		0xC800 2018				Reset Hex Value:				0x00000000																						
Register Description:		Bus activity master.slave																														
Access: Read																																
31															16	15																0
SE	NE	(Reserved)						MPI				PSS				PSN				PMS				PMN								



Register		PMSR		
Bits	Name	Description	Reset Value	Access
31	South Error	When set, the South AHB fields have captured an AHB error and these fields are "stuck" at the first error condition. To clear the error, write a '1' to this bit.	0	RW1C
30	North Error	When set, the North AHB fields have captured an AHB error and these fields are "stuck" at the first error condition. To clear the error, write a '1' to this bit.	0	RW1C
29:2 0	Reserved			
19:1 6	MPI	Indicates which of the ports on the MPI was previously accessing the MCU	0x0	RO
15:1 2	PSS	Indicates which of the Slaves on ARBS was previously accessed the AHBS In the case of an address out-of-range error, this field is 0xF.	0x0	RO
11:8	PSN	Indicates which of the Slaves on ARBN was previously accessed the AHBN In the case of an address out-of-range error, this field is 0xF.	0x0	RO
7:4	PMS	Indicates which of the Masters on ARBS was previously accessing the AHBS	0x0	RO
3:0	PMN	Indicates which of the Master on ARBN was previously accessing the AHBN	0x0	RO

16.7 Event Mapping

There are two connections to the PMU for monitoring the AHB system busses, one is on the North bus and the other on the South bus. [Table 258](#) and [Table 259](#) show the PMU device mapping onto the two busses.

Table 258. AHB North PMU Mapping

PMU Device #	Name	Master	Slave	Retry	Split
0	NPE-A	Y	N	Y	Y
1	NPE-B	Y	N	Y	Y
2	NPE-C	Y	N	Y	Y
3	MCU	N	Y	N	N
4	AHB Bridge	N	Y	N	Y

Table 259. AHB South PMU Mapping

PMU Device #	Name	Master	Slave	Retry	Split
0	XScale BIU	Y	N	Y	Y
1	PCI	Y	Y	Y	N
2	AHB Bridge	Y	N	Y	Y
3	Expansion Bus	Y	Y	Y	Y
4	USBH	Y	Y	Y	N
5	MCU	N	Y	N	N



Table 259. AHB South PMU Mapping

PMU Device #	Name	Master	Slave	Retry	Split
6	APB Bridge	N	Y	N	N
7	AQM	N	Y	N	N
8	RSA	N	Y	N	Y

In the following table, note that there appear to be “too many” entries. For example, there is space allocated for 16 north AHB devices even though there may not be this many defined. To determine the characteristics of a particular model of PMU, one must look at [Table 258](#) and [Table 259](#) to determine which devices are actually present and then in [Table 260](#) to determine the event address in the ESR registers.

Table 260. Event Mux Programming (Sheet 1 of 4)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
MCU Events								
0 Page Miss	MCU[0] Pg0 Misses	MCU[1] Pg1Misses	MCU[2] Pg2 Misses	MCU[3] Pg3 Misses	MCU[4] Pg4 Misses	MCU[5] Pg5 Misses	MCU[6] Pg6 Misses	MCU[7] Pg7 Misses
1 Page Hit	MCU[8] Pg0 Hits	MCU[9] Pg1 Hits	MCU[10] Pg2 Hits	MCU[11] Pg3 Hits	MCU[12] Pg4 Hits	MCU[13] Pg5 Hits	MCU[14] Pg6 Hits	MCU[15] Pg7 Hits
AHB North Device Events								
2	North AHB[0] Grant Occur	North AHB[0] Xfer Occur	North AHB[0] Retry Occur	North AHB[0] Split Occur	North AHB[0] Grant Duration	North AHB[0] Own Duration	North AHB[0] Write Duration	North AHB[0] Read Duration
3	North AHB[1] Grant Occur	North AHB[1] Xfer Occur	North AHB[1] Retry Occur	North AHB[1] Split Occur	North AHB[1] Grant Duration	North AHB[1] Own Duration	North AHB[1] Write Duration	North AHB[1] Read Duration
4	North AHB[2] Grant Occur	North AHB[2] Xfer Occur	North AHB[2] Retry Occur	North AHB[2] Split Occur	North AHB[2] Grant Duration	North AHB[2] Own Duration	North AHB[2] Write Duration	North AHB[2] Read Duration
5	North AHB[3] Grant Occur	North AHB[3] Xfer Occur	North AHB[3] Retry Occur	North AHB[3] Split Occur	North AHB[3] Grant Duration	North AHB[3] Own Duration	North AHB[3] Write Duration	North AHB[3] Read Duration
6	North AHB[4] Grant Occur	North AHB[4] Xfer Occur	North AHB[4] Retry Occur	North AHB[4] Split Occur	North AHB[4] Grant Duration	North AHB[4] Own Duration	North AHB[4] Write Duration	North AHB[4] Read Duration
7	North AHB[5] Grant Occur	North AHB[5] Xfer Occur	North AHB[5] Retry Occur	North AHB[5] Split Occur	North AHB[5] Grant Duration	North AHB[5] Own Duration	North AHB[5] Write Duration	North AHB[5] Read Duration
8	North AHB[6] Grant Occur	North AHB[6] Xfer Occur	North AHB[6] Retry Occur	North AHB[6] Split Occur	North AHB[6] Grant Duration	North AHB[6] Own Duration	North AHB[6] Write Duration	North AHB[6] Read Duration



Table 260. Event Mux Programming (Sheet 2 of 4)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
9	North AHB[7] Grant Occur	North AHB[7] Xfer Occur	North AHB[7] Retry Occur	North AHB[7] Split Occur	North AHB[7] Grant Duration	North AHB[7] Own Duration	North AHB[7] Write Duration	North AHB[7] Read Duration
10	North AHB[8] Grant Occur	North AHB[8] Xfer Occur	North AHB[8] Retry Occur	North AHB[8] Split Occur	North AHB[8] Grant Duration	North AHB[8] Own Duration	North AHB[8] Write Duration	North AHB[8] Read Duration
11	North AHB[9] Grant Occur	North AHB[9] Xfer Occur	North AHB[9] Retry Occur	North AHB[9] Split Occur	North AHB[9] Grant Duration	North AHB[9] Own Duration	North AHB[9] Write Duration	North AHB[9] Read Duration
12	North AHB[10] Grant Occur	North AHB[10] Xfer Occur	North AHB[10] Retry Occur	North AHB[10] Split Occur	North AHB[10] Grant Duration	North AHB[10] Own Duration	North AHB[10] Write Duration	North AHB[10] Read Duration
13	North AHB[11] Grant Occur	North AHB[11] Xfer Occur	North AHB[11] Retry Occur	North AHB[11] Split Occur	North AHB[11] Grant Duration	North AHB[11] Own Duration	North AHB[11] Write Duration	North AHB[11] Read Duration
14	North AHB[12] Grant Occur	North AHB[12] Xfer Occur	North AHB[12] Retry Occur	North AHB[12] Split Occur	North AHB[12] Grant Duration	North AHB[12] Own Duration	North AHB[12] Write Duration	North AHB[12] Read Duration
15	North AHB[13] Grant Occur	North AHB[13] Xfer Occur	North AHB[13] Retry Occur	North AHB[13] Split Occur	North AHB[13] Grant Duration	North AHB[13] Own Duration	North AHB[13] Write Duration	North AHB[13] Read Duration
16	North AHB[14] Grant Occur	North AHB[14] Xfer Occur	North AHB[14] Retry Occur	North AHB[14] Split Occur	North AHB[14] Grant Duration	North AHB[14] Own Duration	North AHB[14] Write Duration	North AHB[14] Read Duration
17	North AHB[15] Grant Occur	North AHB[15] Xfer Occur	North AHB[15] Retry Occur	North AHB[15] Split Occur	North AHB[15] Grant Duration	North AHB[5] Own Duration	North AHB[15] Write Duration	North AHB[15] Read Duration
AHB South Device Events								
18	South AHB[0] Grant Occur	South AHB[0] Xfer Occur	South AHB[0] Retry Occur	South AHB[0] Split Occur	South AHB[0] Grant Duration	South AHB[0] Own Duration	South AHB[0] Write Duration	South AHB[0] Read Duration
19	South AHB[1] Grant Occur	South AHB[1] Xfer Occur	South AHB[1] Retry Occur	South AHB[1] Split Occur	South AHB[1] Grant Duration	South AHB[1] Own Duration	South AHB[1] Write Duration	South AHB[1] Read Duration
20	South AHB[2] Grant Occur	South AHB[2] Xfer Occur	South AHB[2] Retry Occur	South AHB[2] Split Occur	South AHB[2] Grant Duration	South AHB[2] Own Duration	South AHB[2] Write Duration	South AHB[2] Read Duration
21	South AHB[3] Grant Occur	South AHB[3] Xfer Occur	South AHB[3] Retry Occur	South AHB[3] Split Occur	South AHB[3] Grant Duration	South AHB[3] Own Duration	South AHB[3] Write Duration	South AHB[3] Read Duration



Table 260. Event Mux Programming (Sheet 3 of 4)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
22	South AHB[4] Grant Occur	South AHB[4] Xfer Occur	South AHB[4] Retry Occur	South AHB[4] Split Occur	South AHB[4] Grant Duration	South AHB[4] Own Duration	South AHB[4] Write Duration	South AHB[4] Read Duration
23	South AHB[5] Grant Occur	South AHB[5] Xfer Occur	South AHB[5] Retry Occur	South AHB[5] Split Occur	South AHB[5] Grant Duration	South AHB[5] Own Duration	South AHB[5] Write Duration	South AHB[5] Read Duration
24	South AHB[6] Grant Occur	South AHB[6] Xfer Occur	South AHB[6] Retry Occur	South AHB[6] Split Occur	South AHB[6] Grant Duration	South AHB[6] Own Duration	South AHB[6] Write Duration	South AHB[6] Read Duration
25	South AHB[7] Grant Occur	South AHB[7] Xfer Occur	South AHB[7] Retry Occur	South AHB[7] Split Occur	South AHB[7] Grant Duration	South AHB[7] Own Duration	South AHB[7] Write Duration	South AHB[7] Read Duration
26	South AHB[8] Grant Occur	South AHB[8] Xfer Occur	South AHB[8] Retry Occur	South AHB[8] Split Occur	South AHB[8] Grant Duration	South AHB[8] Own Duration	South AHB[8] Write Duration	South AHB[8] Read Duration
27	South AHB[9] Grant Occur	South AHB[9] Xfer Occur	South AHB[9] Retry Occur	South AHB[9] Split Occur	South AHB[9] Grant Duration	South AHB[9] Own Duration	South AHB[9] Write Duration	South AHB[9] Read Duration
28	South AHB[10] Grant Occur	South AHB[10] Xfer Occur	South AHB[10] Retry Occur	South AHB[10] Split Occur	South AHB[10] Grant Duration	South AHB[10] Own Duration	South AHB[10] Write Duration	South AHB[10] Read Duration
29	South AHB[11] Grant Occur	South AHB[11] Xfer Occur	South AHB[11] Retry Occur	South AHB[11] Split Occur	South AHB[11] Grant Duration	South AHB[11] Own Duration	South AHB[11] Write Duration	South AHB[11] Read Duration
30	South AHB[12] Grant Occur	South AHB[12] Xfer Occur	South AHB[12] Retry Occur	South AHB[12] Split Occur	South AHB[12] Grant Duration	South AHB[12] Own Duration	South AHB[12] Write Duration	South AHB[12] Read Duration
31	South AHB[13] Grant Occur	South AHB[13] Xfer Occur	South AHB[13] Retry Occur	South AHB[13] Split Occur	South AHB[13] Grant Duration	South AHB[13] Own Duration	South AHB[13] Write Duration	South AHB[13] Read Duration
32	South AHB[14] Grant Occur	South AHB[14] Xfer Occur	South AHB[14] Retry Occur	South AHB[14] Split Occur	South AHB[14] Grant Duration	South AHB[14] Own Duration	South AHB[14] Write Duration	South AHB[14] Read Duration
33	South AHB[15] Grant Occur	South AHB[15] Xfer Occur	South AHB[15] Retry Occur	South AHB[15] Split Occur	South AHB[15] Grant Duration	South AHB[5] Own Duration	South AHB[15] Write Duration	South AHB[15] Read Duration
AHB Miscellaneous								
34	AHBN Write Duration	AHBN Read Duration	AHBN Idle Duration	AHBN Any Cycle Duration	AHBS Write Duration	AHBS Read Duration	AHBS Idle Duration	AHBN Any Cycle Duration



Table 260. Event Mux Programming (Sheet 4 of 4)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
MPI								
35	MPI[0] Grant Occur	MPI[0] Rd Occur	MPI[0] Wr Occur		MPI[0] Wr Duration	MPI[0] Rd Duration	MPI[0] Latency Duration	MPI[0] Idle Duration
36	MPI[1] Grant Occur	MPI[1] Rd Occur	MPI[1] Wr Occur		MPI[1] Wr Duration	MPI[1] Rd Duration	MPI[1] Latency Duration	MPI[1] Idle Duration
37	MPI[2] Grant Occur	MPI[2] Rd Occur	MPI[2] Wr Occur		MPI[2] Wr Duration	MPI[2] Rd Duration	MPI[2] Latency Duration	MPI[2] Idle Duration
38	MPI[3] Grant Occur	MPI[3] Rd Occur	MPI[3] Wr Occur		MPI[3] Wr Duration	MPI[3] Rd Duration	MPI[3] Latency Duration	MPI[3] Idle Duration
39	Universal Latency 0	Universal Latency 1	Universal Latency 2	Universal Latency 3	Universal Latency 4	Universal Latency 5	Universal Latency 6	Universal Latency 7
40	Universal Latency 8	Universal Latency 9	Universal Latency 10	Universal Latency 11	Universal Latency 12	Universal Latency 13	Universal Latency 14	Universal Latency 15
41	Universal Occurrence 0	Universal Occurrence 1	Universal Occurrence 2	Universal Occurrence 3	Universal Occurrence 4	Universal Occurrence 5	Universal Occurrence 6	Universal Occurrence 7
42	Universal Occurrence 8	Universal Occurrence 9	Universal Occurrence 10	Universal Occurrence 11	Universal Occurrence 12	Universal Occurrence 13	Universal Occurrence 14	Universal Occurrence 15
43- 254								
255	Cycle Count	Cycle Count	Cycle Count	Cycle Count	Cycle Count	Cycle Count	Cycle Count	Cycle Count



17.0 Interrupt Controller

The Intel XScale® Processor supports only a single “IRQ” and “FIQ” interrupt. Expanding the interrupt capabilities beyond that of a single source requires an Interrupt controller external to the Intel XScale processor itself. The APB Interrupt Controller provides IRQ and FIQ interrupts to the Intel XScale processor. The central APB Interrupt Controller accepts all interrupts and depending on the software setting, the interrupts will be passed to the Intel XScale processor.

This version of the interrupt controller extends the number of interrupt sources (as compared to the controller on the Intel® IXP420 Network Processor and Intel® IXP425 Network Processor) as well as introducing a new class of “error interrupts.” The modifications to this unit were designed with the following goals in mind:

- Absolute software compatibility is insured by the fact that no current definitions have been changed, only additional controllability has been added, primarily by uniformly extending the number of interrupts that can be accommodated.
- A new class of interrupts (typically representing error conditions) have unconditional highest priority. An example of this would be some kind of fatal device failure which requires immediate attention.

17.1 Overview

The interrupt controller takes 64 individual interrupts as inputs, refer to [Table 261](#) for configuration and ordering of all the interrupts. These interrupts originate either in internal blocks or from GPIO pins. The controller outputs both an IRQ and an FIQ interrupt to the Intel XScale processor. Any of the 64 input interrupts may be enabled to produce either the IRQ or FIQ output. The INTR_EN/INTR_EN2 Control Register(s) is used to enable an interrupt, and the INTR_SEL/INTR_SEL2 Control Register(s) can be programmed to present an interrupt as an IRQ or an FIQ.

The Intel XScale processor has two methods available to determine which interrupt within the controller caused the interrupt to assert. The status of the entire 64 bit interrupt vector may be read from the pair of registers representing INTR_ST/INTR_ST2. The status of all 64 interrupts enabled as IRQ by INTR_EN/INTR_EN2 may be read from the pair of registers representing INTR_IRQ_ST/INTR_IRQ_ST2. The highest priority interrupt pending on the IRQ may be read by INTR_IRQ_ENC_ST. Note that this register contains an incremented version of the highest priority interrupt, for example, if the highest priority interrupt number was 3, the register, when read, returns 4. This incrementing is due to the fact that the number '0' is used to indicate a spurious or 'no-interrupt' condition. This is also true for the INTR_FIQ_ENC_ST register. Compared to the previous interrupt controller, each of these two vectors is now one bit wider to enable specification of 64 conditions.

Similarly, INTR_FIQ_ST/INTR_FIQ_ST2 contains all interrupts enabled as FIQ while INTR_FIQ_ENC_ST holds the “incremented number” of the highest priority FIQ enabled interrupt pending. Access to all registers is through the APB interface.

It is worth point out that the priority of interrupts only impacts the value in the INTR_IRQ_ENC_ST and INTR_FIQ_ENC_ST registers. To accelerate the software stack, these register encode the highest priority as defined by the priority register and the



positional priority of the interrupts presented to the interrupt controller. Software does not have to abide by this priority, and is free to ignore the encoded status registers. Instead software may read the interrupt status registers directly, and may choose to service interrupts in a different order.

Table 261. Intel XScale® Processor Interrupt Mapping (Sheet 1 of 2)

Interrupt Bit	Default Priority [†]	Source	Description
Int0	0	NPE-A	NPE-A Interrupt (includes Debug/Execution/Error Handling/MBox interrupts)
Int1	1	NPE-B	NPE-B Interrupt (includes Debug/Execution/Error Handling/MBox interrupts)
Int2	2	NPE-C	NPE-C Interrupt (includes Debug/Execution/Error Handling/MBox interrupts)
Int3	3	QM	Queue[1-32]
Int4	4	QM	Queue[33-64]
Int5	5	Timer	General-Purpose Timer[0]
Int6	6	GPIO	GPIO[0]
Int7	7	GPIO	GPIO[1]
Int8	8	PCI	PCI Interrupt
Int9	9	PCI	PCI DMA Channel 1
Int10	10	PCI	PCI DMA Channel 2
Int11	11	Timer	General-Purpose Timer[1]
Int12	12	USB Device	USB device
Int13	13	UART1	UART1 Interrupt
Int14	14	Timer	Timer[2], Time-stamp
Int15	15	UART0	UART0 Interrupt
Int16	16	Timer	Time[3], Watchdog Timer
Int17	17	APB PMU	AHB Performance Monitoring Unit counter rollover
Int18	18	Intel XScale® Processor PMU	Intel XScale processor PMU counter rollover
Int19	19	GPIO	GPIO[2]
Int20	20	GPIO	GPIO[3]
Int21	21	GPIO	GPIO[4]
Int22	22	GPIO	GPIO[5]
Int23	23	GPIO	GPIO[6]
Int24	24	GPIO	GPIO[7]
Int25	25	GPIO	GPIO[8]
Int26	26	GPIO	GPIO[9]
Int27	27	GPIO	GPIO[10]
Int28	28	GPIO	GPIO[11]
Int29	29	GPIO	GPIO[12]
Int30	30	SW Interrupt	SW Interrupt 0
Int31	31	SW Interrupt	SW Interrupt 1
† Priorities of Interrupts 0 through 7 are programmable, 8 through 63 are fixed. Priority 0 is highest priority, 63 is lowest.			

**Table 261. Intel XScale® Processor Interrupt Mapping (Sheet 2 of 2)**

Interrupt Bit	Default Priority [†]	Source	Description
Int32	32	USB Host	USB Host Interrupt
Int33	33	I2C	I2C Interrupt
Int34	34	SSP	SSP Interrupt
Int35	35	TSync	TimeSync Interrupt
Int36	36	EAU	EAU Done Interrupt
Int37	37	SHA	SHA Hashing Done Interrupt
Int38	38	Reserved	Reserved
Int39	39	Reserved	Reserved
Int40	40	Reserved	Reserved
Int41	41	Reserved	Reserved
Int42	42	Reserved	Reserved
Int43	43	Reserved	Reserved
Int44	44	Reserved	Reserved
Int45	45	Reserved	Reserved
Int46	46	Reserved	Reserved
Int47	47	Reserved	Reserved
Int48	48	Reserved	Reserved
Int49	49	Reserved	Reserved
Int50	50	Reserved	Reserved
Int51	51	Reserved	Reserved
Int52	52	Reserved	Reserved
Int53	53	Reserved	Reserved
Int54	54	Reserved	Reserved
Int55	55	Reserved	Reserved
Int56	56	Reserved	Reserved
Int57	57	Reserved	Reserved
Int58	58	SWCP	SWCP Parity Error
Int59	59	Reserved	Reserved
Int60	60	QM	Queue Manager Parity Error
Int61	61	MCU	Multi-Bit ECC Error
Int62	62	EXP	Expansion Bus Parity Error
Int63	63	Reserved	Reserved
† Priorities of Interrupts 0 through 7 are programmable, 8 through 63 are fixed. Priority 0 is highest priority, 63 is lowest.			

17.2 Feature List

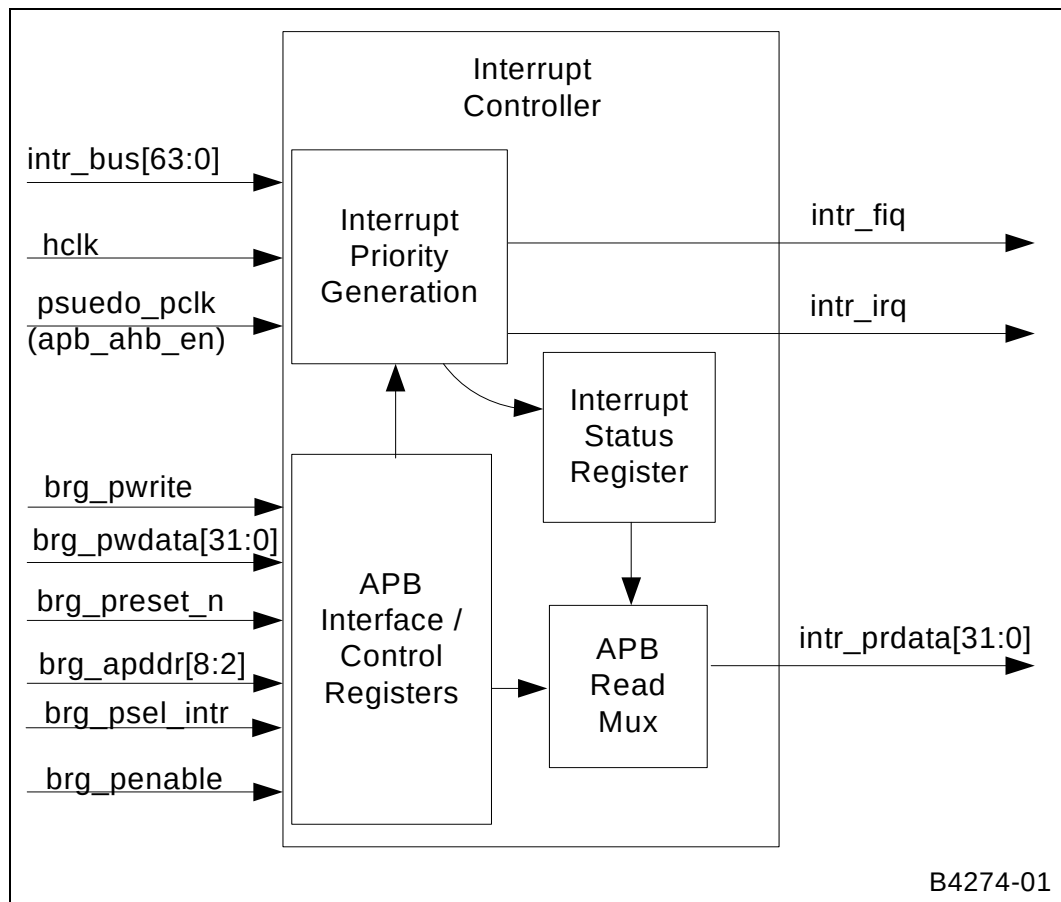
- 64 Supported Interrupts
- Priority encoded servicing of interrupts
- Ability to program the order of the 8 highest priority interrupts.
- Handles pclk- and hclk-generated interrupts

- Standard AMBA APB Interface clocked from qualified pclk.

17.3 Block Diagram

This is the top level block diagram for a module. It typically shows the inputs, outputs and major internal blocks.

Figure 186. Interrupt Controller Block Diagram



17.4 Theory of Operation

The Interrupt Controller takes as inputs 63 individual interrupts (with interrupt 64 reserved). These 63 individual interrupts originate either from internal blocks on the IXP45X/IXP46X network processors or from dedicated GPIO pins. Interrupts are asserted if set ('1') and de-asserted if reset ('0').

There are two classes of interrupts, both of which are defined by their positional priority (e.g position 12 is of higher priority than position 42). The "error" class of interrupts have unconditional priority over "normal" (i.e. not "error" class). The Interrupt Error Enable Register is a 32-bit register that assigns each of the [63:32] interrupts to be the special error interrupts.

For the "normal" class, the highest positional priority interrupt is bit [0] assigned to the NPE A interrupt and the lowest priority interrupt (of the first group) is bit [31] assigned to a Software Interrupt. Interrupts [63:32] are lower priority than the Software



Interrupt to preserve the current functionality, however, for “error” class [63:32] have unconditional priority. The enhancements to the interrupt controller are not intended to alter the connections to the interrupt sources, as the interrupt controller does not influence that mapping, per se. The Interrupt Controller for the IXP45X/IXP46X network processors is implemented to expand the interrupt capabilities of the Intel XScale processor.

The Intel XScale processor has two kinds of interrupts, the first being a special high performance interrupt (FIQ) and a “normal” interrupt (IRQ). (See the Intel XScale processor architecture reference for details about these interrupts and the difference between them.) The interrupts collected by the Interrupt Controller are combined and configured to be an FIQ interrupt or an IRQ. The FIQ signal going to the Intel XScale processor will be set when any of the interrupts assigned to be an FIQ become set. The IRQ signal going to the Intel XScale processor will be set when any of the interrupts assigned to be an IRQ are set.

The interrupt controller does not imply any semantic meaning to the IRQ or FIQ interrupts, only the Intel XScale processor does so. From the perspective of the interrupt controller, there are only two interrupt outputs, and any interrupt source can be funneled to either interrupt. It may be worth pointing out that the FIQ interrupt is not somehow “more important” than the IRQ interrupt, but instead the intent of the Intel XScale processor FIQ interrupt is to provide a low latency interrupt source. The division between IRQ and FIQ is strictly a latency and system software implementation decision.

The Interrupt Controller consists of:

- Two 32-bit interrupt status registers
- Two 32-bit interrupt select registers
- Two 32-bit FIQ status registers
- A 7-bit IRQ highest priority register
- A 32-bit high priority enable register
- Two 32-bit interrupt enable registers
- Two 32-bit IRQ status registers
- A 32-bit interrupt priority registers
- A 7-bit FIQ highest priority register

The Interrupt Controller has no concept of setting or clearing any interrupts and in fact is simply level sensitive *only*. The intent of the Interrupt Controller is to collect and prioritize the received interrupts from other sources. In order to set an interrupt, the device connected to the assigned interrupt line must assert the interrupt.

Clearing of the interrupt must be made at the device that caused the interrupt. If the interrupt is not cleared at the device that asserted the interrupt, the interrupt will be service again. There is a slight delay between resetting the interrupt at the device that caused the interrupt and the resetting of the interrupt due to pipeline delay.

In fact, it can be stated more generally that the interrupt controller has no semantic concept for any of the interrupts. The only semantics associated with the interrupts relate to the choices to funnel an interrupt as either an error interrupt or normal one and as a FIQ or IRQ interrupt.

17.4.1 Interrupt Priority

Selecting the priority of interrupts is done through the assignment of the interrupt priority register and natural interrupt priority assigned by interrupt number. As described, the interrupts follow a natural priority. The interrupt connected to interrupt 0 has the highest priority and the interrupt connected to interrupt 63 has the lowest priority.



In addition to the natural priority the lowest eight interrupts, interrupt 0 to interrupt 7, can be assigned a priority value by writing the interrupt priority register (INTR_PRTY). The interrupt priority register is broken up into eight 3-bit registers.

Bits 0 through 2 of the interrupt priority register assign a priority value to interrupt 0 (NPE A). Bits 3 through 5 of the interrupt priority register assign a priority to interrupt 1 (Ethernet NPE B). The interrupt priority values are assigned in a similar pattern to the assignments above for the first eight interrupts with the last interrupt priority assignment being bits 21 through 23 of the interrupt priority register assigning a priority value to interrupt 7.

The 3-bit interrupt priorities for each of the first eight interrupts can take on a value from 0 to 7. A value of 0 — located the 3-bit interrupt priority register for each of the first eight interrupts — signifies the highest priority interrupt. A value of 7 — located the 3-bit interrupt priority register for each of the first eight interrupts — signifies the lowest priority interrupt. In the case of two interrupts being assigned the same priority, the interrupts natural priority will assign who has the highest priority.

For example, interrupt number 1 and interrupt number 3 both have a value of 0 — written as their 3-bit interrupt priorities. Interrupt number 1 would take priority over interrupt number 3 due to their individual natural priorities.

The priorities assigned to each of the 3-bit interrupt priorities for the first eight interrupts will be set to a value of the corresponding interrupt number (interrupt number 0 gets assigned a value of 0 in the associated 3-bit interrupt priority register, interrupt number 1 gets assigned a value of 1, etc.) when receiving a reset. Therefore, allowing natural priorities to be the default after a reset. While the same effect could be achieved by resetting the register to all zeros (where the positional priority takes precedence) the reset value reflects the *intent* of the priority.

17.4.2 Assigning FIQ or IRQ Interrupts

The Interrupt Controller for the IXP45X/IXP46X network processors provides the capability to assign each interrupt as an FIQ or an IRQ interrupt. As discussed earlier, the Intel XScale processor only receives a single FIQ interrupt signal and a single IRQ interrupt signal.

The Interrupt Controller allows multiple interrupts to be sent to the Intel XScale processor as either FIQ interrupts or IRQ interrupts. Each interrupt may be assigned as an FIQ interrupt or an IRQ interrupt but never both. The interrupts are assigned as an FIQ interrupt or an IRQ interrupt by writing bits in the interrupt select register (INTR_SEL/INTR_SEL2).

The Interrupt Select Register is a pair of 32-bit registers that assigns each of the 64 interrupts to a FIQ interrupt or an IRQ interrupt. Bit 0 of the Interrupt Select Register corresponds to interrupt number 0 (NPE A). Bit 31 of the Interrupt Select Register 2 corresponds to interrupt number 63.

Logic 1 written to a bit in the Interrupt Select Register will assign the corresponding interrupt number as an FIQ interrupt. Writing logic 0 to the same bit in the Interrupt Select Register will assign the corresponding interrupt number as an IRQ interrupt.

For example, the Interrupt Select Register is written with a hexadecimal value of 0x00000005. The result of this write would set interrupt number 0 (NPE A) as an FIQ and interrupt number 2 (Ethernet NPE C) as an FIQ.

All other interrupt numbers would be assigned as IRQ interrupts. All interrupts are assigned as IRQ interrupts upon receiving a reset because the register is cleared to 0x00000000.



The Interrupt Error Enable Register is a 32-bit register that assigns each of the [63:32] interrupts to be the special error interrupts. These interrupts take unconditional priority over all other interrupts. Logic 1 written to a bit in the Interrupt Error Enable Register will imply that the interrupt will take unconditional priority over all other interrupts. A Logic 0 written to the same bit will imply that the interrupt has a simple positional priority.

17.4.3 Enabling and Disabling Interrupts

The interrupts on the IXP45X/IXP46X network processors can be individually enabled or disabled by writing to the pair of Interrupt-Enable Registers (INTR_EN/INTR_EN2). By disabling an interrupt, the Intel XScale processor will not be interrupted by either the FIQ interrupt signal or the IRQ interrupt signal when an interrupt occurs on the corresponding disabled interrupt input.

For instance, interrupt number 0 is disabled and an interrupt occurs on interrupt number 0. The interrupt generated by interrupt number 0 will not be seen by the Intel XScale processor.

The Interrupt-Enable Register is a pair of 32-bit registers that can individually enable or disable each of the 64 interrupts. Bit 0 of the Interrupt-Enable Register corresponds to interrupt number 0 (NPE A). Bit 31 of the Interrupt-Enable Register 2 corresponds to interrupt number 63.

Logic 1 written to a bit in the Interrupt-Enable Register will enable the corresponding interrupt number. Writing logic 0 to the same bit in the Interrupt-Enable Register will disable the corresponding interrupt number.

For example, the Interrupt-Enable Register is written with a hexadecimal value of 0x0000000A. The result of this write would enable interrupt number 1 (Ethernet NPE B) and interrupt number 3 (Queue Manager Queues 1-32). All other interrupt numbers would be disabled. All interrupts are disabled upon receiving a reset because the register is cleared to 0x00000000.

17.4.4 Reading Interrupt Status

The IXP45X/IXP46X network processors provide several mechanisms in which interrupt status can be obtained from the Interrupt Controller. One method of obtaining interrupt status is to read the interrupt status register directly (INTR_ST/INTR_ST2).

The Interrupt Status Register is a pair of 32-bit registers that has a one-for-one relationship with the interrupt number. Interrupt number 0 will be the status represented on bit 0 of the Interrupt Status Register and interrupt number 31 will be the status represented on bit 31 of the Interrupt Status Register. Interrupt numbers 32 to 63 are represented in the second status register.

Reading Logic 1 from a bit in the Interrupt Status Register represent that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller. For example, a read is performed on the Interrupt Status Register and the result returned is a hexadecimal 0x00000002. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 1 (Ethernet NPE B) has caused an interrupt.

The Intel XScale processor will service the interrupt and clear the interrupt by updating the register in the device that caused the interrupt condition (in this example, in the Ethernet NPE B). To clear this specific interrupt will require updating registers in the NPE A, not this interrupt controller. Using the Interrupt Status Register allows an interrupt service routine to assign relative priorities to the interrupts and removes all



relationship from the priority algorithms or the interrupt types assigned by the Interrupt Controller for the IXP45X/IXP46X network processors. The Interrupt Status Register is set to all zeros upon reset.

All other methods of reading interrupt status will involve the use of the register sets provided by the IXP45X/IXP46X network processors to aid in the determination of which interrupts should be serviced first.

The IXP45X/IXP46X network processors provide the capability of reading the interrupt status of the interrupt numbers that have been assigned as FIQ interrupts or reading the interrupt status of the interrupt numbers that have been assigned as IRQ interrupts. The status of the interrupt numbers that have been assigned as FIQ interrupts can be read by reading the FIQ status register (INTR_FIQ_ST/INTR_FIQ_ST2). The status of the interrupt numbers that have been assigned as IRQ interrupts can be read by reading the IRQ status register (INTR_IRQ_ST/INTR_IRQ_ST2).

The FIQ Status Register and the IRQ Status Register are 32-bit registers that have a one-for-one relationship with the interrupt number. Interrupt number 0 (NPE A) will be the status represented on bit 0 of both the FIQ Status Register and the IRQ Status Registers. Interrupt number 31 will be the status represented on bit 31 of both the FIQ Status Register and the IRQ Status Registers. Interrupt numbers 32 to 63 are represented in the second status registers for both the IRQ and the FIQ.

Reading logic 1 from a bit in either the FIQ Status Register(s) or the IRQ Status Register(s) represent that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller and that the interrupt is enabled for the FIQ or IRQ interrupts respectively. For example, a read is performed on the FIQ Status Register and the result returned is a hexadecimal 0x00000001. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 0 (NPE A) has caused an interrupt and the interrupt is enabled as a FIQ interrupt.

The Intel XScale processor will service the interrupt and clear the interrupt by updating the register that caused the interrupt condition in the NPE A. The same action will be applied to an interrupt that would be caused by an IRQ interrupt. Allowing the capability to separate the FIQ and IRQ interrupts allows separate interrupt service routines to be built based on the type of interrupt received. This allows greater control in applications that may be developed for the IXP45X/IXP46X network processors.

The FIQ Status Registers and IRQ Status Registers are set to all zeros upon reset.

The IXP45X/IXP46X network processors also allow the capability to read the highest-priority IRQ interrupt or the highest-priority FIQ interrupt as determined by the priority algorithm described in [“Interrupt Priority” on page 807](#).

The highest-priority IRQ interrupt can be read by reading the IRQ Highest-Priority Register (INTR_IRQ_ENC_ST). The highest-priority FIQ interrupt can be read by reading the FIQ Highest-Priority Register (INTR_FIQ_ENC_ST). The IRQ Highest-Priority Register and the FIQ Highest-Priority Registers are 7-bit registers that will return the highest-priority interrupt number (i.e. its position) of each the IRQ interrupts and the FIQ interrupts. Note that the value of the encoding status registers is always the positional value, and this is true even if the lower bits are programmed in a different *priority* order. For example, if position 0 is actually set to priority 7 via the INTR_PRTY register, and it is the highest *priority* interrupt (because positions 7:1 are not interrupting) the value of the encoding status register will be zero, because that is the position of the interrupt, not the value 7 which is its priority.

The value obtained by reading the IRQ Highest-Priority Register will be the interrupt number of the highest-priority IRQ interrupt, incremented by one and the sum of the add left shifted by two bits. Therefore, the seven bits — that contain the IRQ highest priority — are actually located in bits 2 through 8 of the IRQ Highest-Priority Register.



For example, interrupt number 1 (Ethernet NPE B) is the highest-priority IRQ interrupt, the value obtained when reading the IRQ Highest-Priority Interrupt Register would be hexadecimal 0x00000008. A value of 0 — returned when reading the IRQ Highest-Priority Register — signifies that no IRQ interrupts are pending.

The IRQ Highest-Priority Register will be reset to a value of 0. The FIQ Highest-Priority Register will behave in an identical fashion to the IRQ Highest-Priority Register.

17.5 Error Enable Register

The ERROR_EN2 register is what defines the difference between the two so-called classes of interrupts. Interrupts [31:0] are by definition (for compatibility reasons) never of the error class. Only the new interrupts [63:32] can be in this class. The purpose of the error class of interrupts is to force the servicing of error conditions above normal conditions because they usually represent some kind of failure condition. For example, NPE A may have parity bits added to its data memory. If a parity error is detected an interrupt is generated. However if that interrupt's error enable bit is TRUE, the parity error takes unconditional priority over the "normal" positional priority interrupts. Note that this does not alter the value of the number reported in the encoding status registers, only that the priority scheme is altered.

The bit in ERROR_EN2[0] corresponds to interrupt 32 and the bit in ERROR_EN2[31] corresponds to interrupt 63. If the interrupt is enabled, the error enable is TRUE, and the interrupt becomes active, that interrupt takes priority over all other interrupts. If more than one error class of interrupts is being reported, the highest positional priority is reported in the encoding status register assigned to that interrupt (i.e. FIQ or IRQ).

17.6 Interrupt Controller Register Descriptions

Table 262. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 263. Interrupt Controller Memory Mapped Registers (Sheet 1 of 2)

Address	Access	Name	Description
0xC8003000	RO	INTR_ST	Interrupt Status Register
0xC8003004	RW	INTR_EN	Interrupt Enable Register
0xC8003008	RW	INTR_SEL	Interrupt Select Register
0xC8003020	RO	INTR_ST2	Interrupt Status Register 2
0xC8003024	RW	INTR_EN2	Interrupt Enable Register 2
0xC8003028	RW	INTR_SEL2	Interrupt Select Register 2
0xC800302C	RO	INTR_IRQ_ST2	IRQ Status register 2
0xC8003030	RO	INTR_FIQ_ST2	FIQ status Register 2
0xC8003034	RW	ERROR_EN2	Error Priority Enable Register



Table 263. Interrupt Controller Memory Mapped Registers (Sheet 2 of 2)

Address	Access	Name	Description
0xC800300C	RO	INTR_IRQ_ST	IRQ Status register
0xC8003010	RO	INTR_FIQ_ST	FIQ status Register
0xC8003014	RW	INTR_PRTY	Interrupt Priority Register
0xC8003018	RO	INTR_IRQ_ENC_ST	IRQ Highest Priority Register
0xC800301C	RO	INTR_FIQ_ENC_ST	FIQ Highest Priority Register
0xC8003020	RO	INTR_ST2	Interrupt Status Register 2
0xC8003024	RW	INTR_EN2	Interrupt Enable Register 2
0xC8003028	RW	INTR_SEL2	Interrupt Select Register 2
0xC800302C	RO	INTR_IRQ_ST2	IRQ Status register 2
0xC8003030	RO	INTR_FIQ_ST2	FIQ status Register 2
0xC8003034	RW	ERROR_EN2	Error Priority Enable Register

17.6.1 Interrupt Status Register

Register Name:		INTR_ST																			
Physical Address:		0xC800 3000								Reset Hex Value:											
Register Description:		This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but simply provides visibility into the individual interrupt outputs from each source. For this reason, there is no “reset value”.																			
Access: Read.																					
31																				0	
Incoming Interrupt Status [31:0]																					

Register Name:		INTR_ST2															
Physical Address:		0xC800 3020								Reset Hex Value:							
Register Description:		This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but simply provides visibility into the individual interrupt outputs from each source. For this reason, there is no “reset value”.															
Access: Read.																	
31																	0
Incoming Interrupt Status [63:32]																	



17.6.2 Interrupt Enable Register

Register Name:		INTR_EN																											
Physical Address:		0xC800 3004								Reset Hex Value:		0x00000000																	
Register Description:		Provides enables for the interrupts. This register allows the Intel XScale® Processor to disable interrupts from selected blocks. To enable an interrupt, a 1 is written into corresponding bit, to disable it, a 0 is written.																											
Access: Read/Write.																													
31																													0
Interrupt Enables[31:0]																													

Register Name:		INTR_EN2																											
Physical Address:		0xC800 3024												Reset Hex Value:		0x00000000													
Register Description:		Provides enables for the interrupts. This register allows the Intel XScale® Processor to disable interrupts from selected blocks. To enable an interrupt, a 1 is written into corresponding bit, to disable it, a 0 is written.																											
Access: Read/Write.																													
31																													0
Interrupt Enables[63:32]																													

17.6.3 Interrupt Select Register

Register Name:		INTR_SEL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
Physical Address:		0xC800 3008												Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
Register Description:		This register decides if an interrupt is to be presented to the Intel XScale® Processor as an FIQ or an IRQ. If a bit corresponding to an interrupt is set (to 1), that interrupt is presented as a FIQ. If the bit is reset to 0, the interrupt is presented as an IRQ.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
Access: Read/Write.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
31																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													

Register Name:		INTR_SEL2																											
Physical Address:		0xC800 3028								Reset Hex Value:		0x00000000																	
Register Description:		This register decides if an interrupt is to be presented to the Intel XScale® Processor as an FIQ or an IRQ. If a bit corresponding to an interrupt is set (to 1), that interrupt is presented as a FIQ. If the bit is reset to 0, the interrupt is presented as an IRQ.																											
Access: Read/Write.																													
31																													0
Interrupt Selects[63:32]																													



17.6.4 IRQ Status Register

Register Name:		INTR_ IRQ_ST																											
Physical Address:		0xC800 300C												Reset Hex Value:		0x00000000													
Register Description:		This register is an “AND” of the incoming status with the INTR_EN and the inverted version of the INTR_SEL. The INTR_IRQ_ST indicates which of the incoming interrupts are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled.																											
Access: Read.																													
31																													0
IRQ Status Information[31:0]																													

Register Name:		INTR_ IRQ_ST2																											
Physical Address:		0xC800 302C								Reset Hex Value:		0x00000000																	
Register Description:		This register is an “AND” of the incoming status with the INTR_EN2 and the inverted version of the INTR_SEL2. The INTR_IRQ_ST2 indicates which of the incoming interrupts are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled.																											
Access: Read.																													
31																													0
IRQ Status Information[63:32]																													

17.6.5 FIQ Status Register

Register Name:		INTR_FIQ_ST																													
Physical Address:		0xC800 3010								Reset Hex Value:		0x00000000																			
Register Description:		This register is an “AND” of the incoming status with the INTR_EN and the INTR_SEL. The INTR_FIQ_ST indicates which of the incoming interrupts are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled.																													
Access: Read.																															
31																															0
FIQ Status Info[31:0]																															

Register Name:		INTR_FIQ_ST2																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
Physical Address:		0xC800 3030												Reset Hex Value:		0x00000000																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
Register Description:		This register is an “AND” of the incoming status with the INTR_EN2 and the INTR_SEL2. The INTR_FIQ_ST indicates which of the incoming interrupts are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
Access: Read.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
31																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				



17.6.6 Interrupt Priority Register

Register Name:		INTR_PRTY															
Physical Address:		0xC800 3014								Reset Hex Value:		0x00FAC688					
Register Description:		The highest eight priority interrupts can be programmed via this register, each of the 3-bit sets can be programmed to any priority from 0(000) through 7(111). This register applies to both IRQ and FIQ interrupts.															
Access: Read/Write.																	
31							24	23									0
Reserved						Interrupt Priority selects											

Register		INTR_PRTY		
Bits	Name	Description	Reset Value	Access
31:24	Reserved	Reserved, Read as undefined, write as 0	0x00	RW
23:21	Prior_Intbus7 [2:0]	Set the priority of the Intr_bus [7]; default is 7	111	RW
20:18	Prior_Intbus6 [2:0]	Set the priority of the Intr_bus [6]; default is 6	110	RW
17:15	Prior_Intbus5 [2:0]	Set the priority of the Intr_bus [5]; default is 5	101	RW
14:12	Prior_Intbus4 [2:0]	Set the priority of the Intr_bus [4]; default is 4	100	RW
11:9	Prior_Intbus3 [2:0]	Set the priority of the Intr_bus [3]; default is 3	011	RW
8:6	Prior_Intbus2 [2:0]	Set the priority of the Intr_bus [2]; default is 2	010	RW
5:3	Prior_Intbus1 [2:0]	Set the priority of the Intr_bus [1]; default is 1	001	RW
2:0	Prior_Intbus0 [2:0]	Set the priority of the Intr_bus [0]; default is 0	000	RW

17.6.7 IRQ Highest-Priority Register

Register Name:		INTR_IRQ_ENC_ST																																						
Physical Address:		0xC800 3018										Reset Hex Value:		0x00000000																										
Register Description:		This register returns the “incremented number” of the highest-priority interrupt that is pending for the IRQ. For example, if interrupt 0 is the highest IRQ pending, the register returns 1. If the register returns 0, it means that there is no interrupt pending or a spurious interrupt. Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read. This allows the register’s contents to be directly used as an offset into a jump table for interrupt vectoring.																																						
		Access: Read.																																						
31																												8								2		1		0
Reserved																				IRQ_ENC_ST								Rsvd												

Register		INTR_IRQ_ENC_ST		
Bits	Name	Description	Reset Value	Access
31:9	Reserved	Reserved, Read as undefined, write as 0	0x000000	RO
8:2	IRQ_ENC_ST	Indicates the highest priority pending interrupt (the interrupt “number” incremented by 1)	0x00	RO
1:0	Reserved	Reserved, Read as undefined, write as 0	0x0	RO



17.6.8 FIQ Highest-Priority Register

Register Name:		INTR_FIQ_ENC_ST																											
Physical Address:		0xC800 301C								Reset Hex Value:		0x00000000																	
Register Description:		This register returns the “incremented number” of the highest-priority interrupt that is pending for the FIQ. For example, if interrupt ‘0’ is the highest FIQ pending, the register returns 1.																											
		Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read. This allows the register’s contents to be directly used as an offset into a jump table for interrupt vectoring.																											
Access: Read.																													
31																8									2		1		0
Reserved													FRQ_ENC_ST					Rsvd											

Register		INTR_FIQ_ENC_ST		
Bits	Name	Description	Reset Value	Access
31:9	Reserved	Reserved, Read as undefined, write as 0	0x000000	RO
8:2	FIQ_ENC_ST	Indicates the highest-priority, pending fast interrupt (the interrupt “number” incremented by 1)	0x00	RO
1:0	Reserved	Reserved, Read as undefined, write as 0	0x0	RO

The additional bit defined (i.e. bit 8) of the highest priority register is in the same bit position as the extended register address. The higher bit of the status can be used as part of the address calculation to determine which group of registers to access.

17.6.9 Error High Priority Enable Register

Register Name:		ERROR_EN2																	
Physical Address:		0xC800 3034								Reset Hex Value:		0x00000000							
Register Description:		This register decides if an interrupt is to be presented to the Intel XScale® Processor above all other priorities. This affects the priority only, not the number reported in the *_ENC_ST registers. Bit 0 of this register corresponds to interrupt 32 and bit 31 corresponds to interrupt 63.																	
Access: Read/Write.																			
31																			0
Error High Priority Enable[63:32]																			

§ §



18.0 Operating System Timer

18.1 Overview

The OST serves the function of a watchdog timer as well as a general-purpose timer and is capable of generating interrupts at predetermined intervals. The module contains several registers which are written and read via the APB interface. All four timers can generate interrupts, that can be independently cleared.

All timers in the OST run off the AMBA APB clock by default. An additional feature to the OST makes it possible for each timer to use a prescaled clock. The timers may also enable a 3/4 scaler to allow the emulation of a 50-MHz clock from a APB 66.667-MHz input clock. All accesses to the OST are 32 bits (word) wide. Bits that correspond to reserved register bits are ignored on “writes.” These same bits will return zero on “reads.”

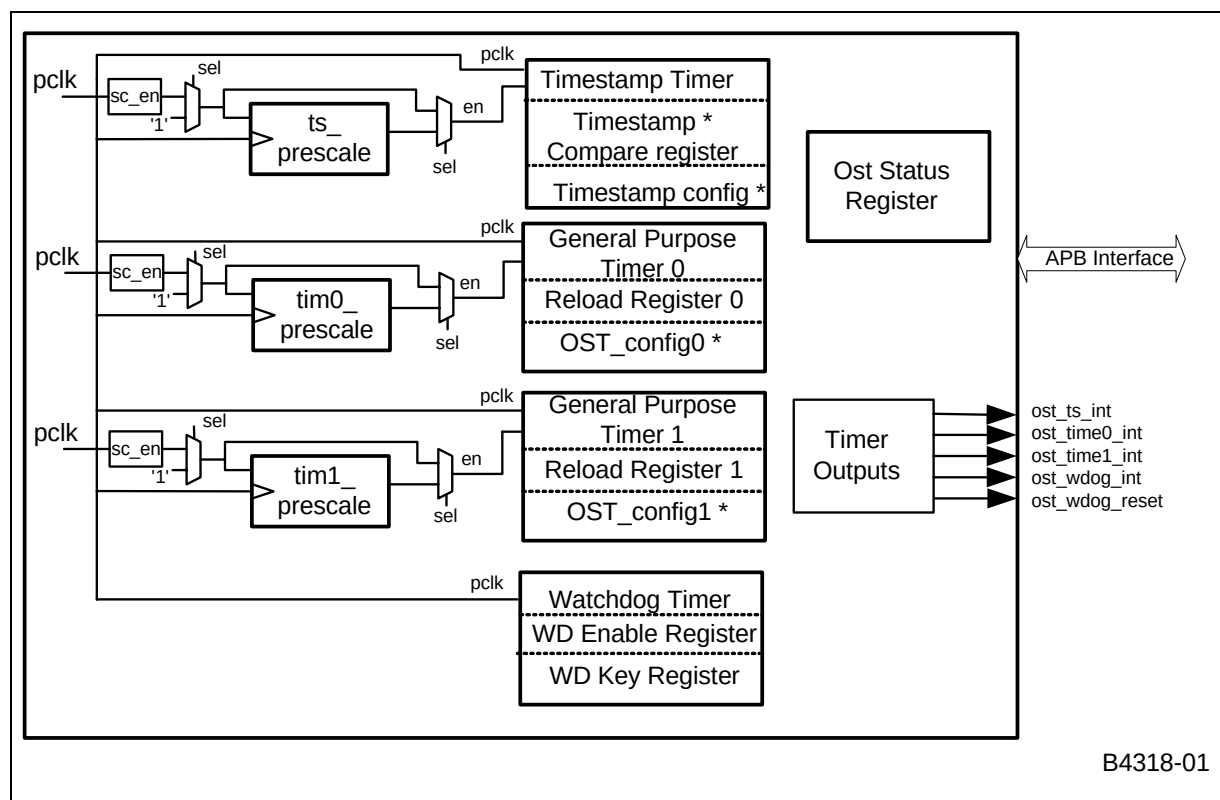
18.2 Feature List

- Two readable decrementing general-purpose timers with read/writable reload registers (each suitable for use as an OS timer)
- 32-bit timestamp incrementing timer, with timestamp compare register.
- 32-bit watchdog timer (decrementing)
- Interrupt, reset enable, and watchdog count for watchdog timer
- “Key” protected access to watch dog register
- A 1-bit, warm-reset register that signifies when a warm reset has occurred from this unit
- Peripheral on the AMBA APB
- Each timer except the watchdog timer can enable a 16-bit Prescaler.
- Each timer except the watchdog can enable a 3/4 scale function (to emulate a 20-ns clock [50 MHz] off a 66.66667-MHz clock).
- Separate configuration registers for each of the timers.
- The timestamp timer and the two general-purpose counters can be paused only if the prescaler or the 3/4 scale function on the respective timer is enabled.
- Compatible with Intel® IXP400 Software.

18.3 Block Diagram

A block diagram of the OST unit is shown in [Figure 187](#).

Figure 187. Operating System Timer Block Diagram



18.4 Theory of Operation

The OST supports four timers, the first is a watchdog timer, the second a free-running timestamp timer and the last two are general-purpose timers capable of generating interrupts at predetermined intervals.

All of the registers/counters operate on the APB bus clock by default. By writing a value into separate configuration and prescale registers, a prescaled enable can be used for each timer. Each timer, except for watchdog timer, can also enable a 3/4 scale function to emulate a 20-ns clock from a 66.667-MHz input clock. The 3/4 scaler can also be used in combination with the prescaler.

18.4.1 Watchdog Timer Operation

The watchdog timer is used by the software to monitor inactivity. The watchdog timer is composed of four components:

- A 32-bit writable down counter — (ost_wdog)
- A 3-bit enable register — (ost_wdog_enab)
- A 16-bit key register — (ost_wdog_key)
- A 5-bit status register — (ost_sts)

The ost_wdog_key register can be written at any time over the APB bus. However, all the watch-dog registers can only be written to when the ost_wdog_key register contains the value 0x482E (this value referred to from here on as “key-value”). A write



to these registers has no effect unless `ost_wdog_key = key_value`. This is to prevent accidental writes to these registers. The typical operation would be for the software to write the key-value into the `ost_wdog_key` register, write to the `ost_wdog` and/or `ost_wdog_enab`, then the software would write a value other than key-value into the `ost_wdog_key` register. It will then periodically (before the `ost_wdog` reaches zero) write the key-value into the `ost_wdog_key` register, write a new count value into the `ost_wdog`, then write a value other than the key-value into the `ost_wdog_key` register.

Upon reset the `ost_wdog` register is set to all ones, the `ost_wdog_enab` register is disabled.

The `ost_wdog_enab` register contains three bits: `wdog_cnt_ena`, `wdog_int_ena` and `wdog_reset_n_ena`. When enabled (by the `wdog_cnt_ena` bit) the `ost_wdog` decrements. If it reaches zero it stops decrementing and drives a `ost_wdog_int` signal and/or a `ost_wdog_reset_n` signal, depending on the values of `wdog_int_ena` and `wdog_reset_n_ena`. These signals will continue to be driven active while the count in the `ost_wdog` equals zero. If the `ost_rst_enb` bit is set, the entire chip will be reset which will have the effect of setting all the OST registers to their initial state values as described in the previous paragraph. If the `wdog_rst_ena` bit is not set a reset will not occur. In this case the `ost_wdog` will remain zeros until software writes a different value into it.

If the `ost_wdog` ever reaches zero when `wdog_rst_en` is enabled, the `warm_reset` register bit is set. This register bit can be cleared by writing a '1' to it or by the assertion of `reset_cold_n` which is derived from the Power On Reset.

18.4.2 Timestamp Timer Operation

The timestamp timer is a readable 32-bit free-running counter. Upon reset it is by default set to `32'h0000_0001` and starts counting up as soon as reset is released. Unless the timestamp compare register value is different from `32'h0000_0000`, the timer rolls over and generates an interrupt signal. The `ost_ts_int` signal is asserted until cleared by a writing a 1'b1 to the appropriate bit in `ost_sts` register. If the compare register has a different value than `32'h0000_0000`, the timer generates an interrupt when the timestamp counter equals the compare register value, the counter continues to run. The `ost_ts_int` signal will stay asserted until cleared by writing a 1'b1 to the appropriate bit in the `ost_sts` register. A prescale is activated by writing a value other than `16'h0000` into the prescale value of the timestamp prescale register. The 3/4 scale_en is activated by writing a 1'b1 to `ts_scale_en` in the timestamp configuration register. The timer can be paused by setting the `ts_pause_en` bit in the timestamp configuration register. The timer can be restarted after a pause by clearing the `ts_pause_en` bit. The Timestamp timer can only use the pause/restart feature if prescale or 3/4scale_en is activated. When using the prescaler, the prescale register needs updated before the counter is restarted to ensure the correct division rate. No extra action is necessary if only the 3/4 scale is used (`ts_scale_en` is 1'b1 and `ost_ts_pre = 16'h0000`).

Example:

Assume the `ost_ts_pre` register holds the value `16'h0001`, and `ts_scale_en` is 1'b0. This means the timestamp is counting with a APB clk divided by 2.

To pause the timer.

1. Set `ts_pause_en` to 1'b1.

When desirable to let the counter continue:

2. Refresh the `ost_ts_pre` register with a `16'h0001`, then clear `ts_pause_en`.



18.4.3 General-Purpose Timers Operation

There are two 32-bit general-purpose timers. The general-purpose timers have six components:

- A 32-bit down-counter
- A one_shot control bit
- A cnt_enable control bit
- A 32-bit reload value register
- A 16-bit prescale register
- A 4-bit configuration register

The purpose of the timers are to generate timed or periodic interrupts to the Intel XScale processor.

Upon reset all the registers are initialized to zero.

The 32-bit reload value register will be packed into the same APB addressable word with the one_shot control bit and the cnt_enable control bit. The reload value occupying bits 31-2, one_shot in bit 1 and cnt_enable in bit 0. To achieve higher granularity the two least significant bits are fetched from bit 0 and bit 1 in the General-Purpose Configuration register.

For predictable operation, stop the timer before writing a new value into the reload register.

The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, and the counter reaches zero, the counter will load the “reload value register” and start counting down. If the one_shot control register is set to one it will load the “reload value register” and stop counting.

The cnt_enable control bit specifies whether the counter is enabled for counting. When set to one the counter is enabled.

The timer simply counts down when the enable bit is set to one. The ost_tim*_int signal is asserted when the count reaches zero and remains asserted until cleared by a writing a 1'b1 to the appropriate bit in ost_sts register.

The General-Purpose Timers have the option of using the prescaled clock, and/or the 3/4 scale_enable logic. The value of prescale is contained in the general-purpose prescale registers. When and only when a prescaled clock is used, users have an option to pause/restart the timers. This option can be controlled in the General-Purpose Configuration registers. By writing a 1'b1 to tim0_pause_en or tim1_pause_en, the respective counter will stop. To “wake up” the counter; refresh the prescale register for the respective counter by re-writing the same prescale value to the associated prescale register ost_timx_pre, and then clear the respective timx_pause_en.

Example on how to operate the General-Purpose Timer0.

1. ost_tim0_cfg <= 0x07 Configures the timer with 3/4 scale_en and the two lsb bits in the reload value to be ones.
2. ost_tim0_pre <= 0x19 Loads the prescaler with 25, which means a divide by 26.
3. ost_tim0_rl <= 0x15 Sets the reload value to be 0x14 and the lsb enables the counter. The reload value will now be 0x14 plus 0x03, set in step 1, a total of 0x17 (23). Since 0 is included in decrementing the real value is N+1 = 24. This setup will now produce an interrupt interval of

$$I_{\text{period}} = 3/4\text{scale} * \text{prescale\_value} * \text{reload\_value}$$



$$= 20\text{ns} * 26 * 24 = 12.48\mu\text{s}$$

If it is desirable to pause the timer, then use the tim0_pause_en to halt counting

4. ost_tim0_cfg <= 0x0F This will set tim0_pause_en

To enable the counting such that the counter continues from its paused value

5. ost_tim0_pre <= 0x19 This is a refresh write that is necessary for the prescale to properly work.

6. ost_tim0_cfg <= 0x07 Clear tim0_pause_en

If no prescaler is used, step 5 is not necessary to do.

18.4.4 Clock Prescale

Each timer except the watch dog is provided with a programmable prescaler that divides the input clock with a 16-bit value. The input clock can be either the APB system clock or a 20-ns version of it (3/4scale). By default all timers use the apb clock. The 16-bit prescale value ranges from divide by 2 to 65,536, and results in a new clock enable available for the timers that ranges from 33.33 MHz down to 1017.26 Hz, the divider is a divide by N+1, where N = {0,1,2,3...}. The value for the new available clock enable is written in the internal register for the desired timer, the new value is updated on the next positive edge of pclk.

18.5 Detailed Register Descriptions

The registers are accessible via the APB bus interface.

Table 264. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 265. Register Summary

Address	Register Name	Description	Reset Value	Attribute	Counter type
"0xC8005000"	"ost_ts"	"Timestamp timer"	"0x00000001"	RW	Up
"0xC8005004"	"ost_tim0"	"General-Purpose Timer 0"	"0x00000000"	RO	Down
"0xC8005008"	"ost_tim0_rl"	"General-Purpose Timer 0 Reload"	"0x00000000"	RW	n/a
"0xC800500C"	"ost_tim1"	"General-Purpose Timer 1"	"0x00000000"	RO	Down
"0xC8005010"	"ost_tim1_rl"	"General-Purpose Timer 1 Reload"	"0x00000000"	RW	n/a
"0xC8005014"	"ost_wdog"	"Watchdog Counter"	"0xFFFFFFFF"	RW [†]	Down
"0xC8005018"	"ost_wdog_enab"	"Watchdog Enable Register"	"0x00000000"	RW [†]	n/a
"0xC800501C"	"ost_wdog_key"	"Watchdog Key Register"	"0x00000000"	RW	n/a
"0xC8005020"	"ost_status"	"Timer Status Register"	"0x00000000"	RW1C	n/a

[†] Only through watchdog protection mechanism

**Table 265. Register Summary**

Address	Register Name	Description	Reset Value	Attribute	Counter type
"0xC8005024"	"ost_ts_cmp"	"Timestamp Compare Register"	"0x00000000"	RW	n/a
"0xC8005028"	"ost_ts_cfg"	"Timestamp Configuration Register"	"0x00000000"	RW	n/a
"0xC800502C"	"ost_ts_pre"	"Timestamp Prescale Register"	"0x00000000"	RW	n/a
"0xC8005030"	"ost_tim0_cfg"	"General-Purpose Timer 0 Configuration"	"0x00000000"	RW	n/a
"0xC8005034"	"ost_tim0_pre"	"General-Purpose Timer 0 Prescale Register"	"0x00000000"	RW	n/a
"0xC8005038"	"ost_tim1_cfg"	"General-Purpose Timer 1 Configuration"	"0x00000000"	RW	n/a
"0xC800503C"	"ost_tim1_pre"	"General-Purpose Timer 1 Prescale Register"	"0x00000000"	RW	n/a

† Only through watchdog protection mechanism

18.5.1 Timestamp Timer

Register Name:		ost_ts																																			
Physical Address:		0xC8005000				Reset Hex Value:				0x00000001																											
Register Description:		Timestamp timer																																			
Access: Read/Write																																					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
timer_val																																					

Register		ost_ts			
Bits	Name	Description	Reset Value	Access	
31:0	timer_val	Current value of the timer.	32'd0	RW	

18.5.2 General-Purpose Timer 0

Register Name:		ost_tim0																													
Physical Address:		0xC8005004				Reset Hex Value:				0x00000000																					
Register Description:		General-Purpose Timer 0																													
Access: Read																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
timer_val																															

Register		ost_tim0			
Bits	Name	Description	Reset Value	Access	
31:0	timer_val	Current value of the timer.	32'd0	RO	



18.5.3 General-Purpose Timer 0 Reload

Register Name:		ost_tim0_rl																													
Physical Address:		0xC8005008								Reset Hex Value:						0x00000000															
Register Description:		General-Purpose Timer 0 Reload																													
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reload_val																												one_shot	enable		

Register		ost_tim0_rl			
Bits	Name	Description	Reset Value	Access	
31:2	reload_val	Reload value. Value to be reloaded into ost_tim0. Bits 1 and 0 will be fetched from ost_tim0_cfg. For predictable operation, stop the timer before writing a new value into the reload register. Note: As the counter counts to zero, writing a value of X to these bits will generate interrupts at intervals of (X*4)+1 cycles, unless the two least significant bits in ost_tim0_cfg are filled.	30'd0	RW	
1	tim1_one_shot	The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, when the counter reaches zero it will load the "reload value register" and start counting down. If the one_shot control register is set to one it will load the "reload value register" and stop counting.	0	RW	
0	tim0_enable	The cnt_enable register controls whether the counter is enabled for counting. When set to one the counter is enabled.	0	RW	

18.5.4 General-Purpose Timer 1

Register Name:		ost_tim1																													
Physical Address:		0xC800500C										Reset Hex Value:				0x00000000															
Register Description:		General-Purpose Timer 1																													
Access: Read																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
timer_val																															

Register		ost_tim1			
Bits	Name	Description	Reset Value	Access	
31:0	timer_val	Current value of the timer	32'd0	RO	



18.5.5 General-Purpose Timer 1 Reload

Register Name:								ost_tim1_rl																							
Physical Address:								0xC8005010								Reset Hex Value:								0x00000000							
Register Description:								General-Purpose Timer 1 Reload																							
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reload_val																										one_shot	enable				

Register		ost_tim1_rl		
Bits	Name	Description	Reset Value	Access
31:2	reload_val	Reload value. Value to be reloaded into ost_tim1. Bits 1 and 0 will be fetched from ost_tim1_cfg. For predictable operation, stop the timer before writing a new value into the reload register. Note: As the counter counts to zero, writing a value of X to these bits will generate interrupts at intervals of (X*4)+1 cycles, Unless the two least significant bits in ost_tim0_cfg is filled.	30'd0	RW
1	tim1_one_shot	The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, when the counter reaches zero will load the "reload value register" and start counting down. If the one_shot control register is set to one it will load the "reload value register" and stop counting.	0	RW
0	tim1_enable	The cnt_enable register controls whether the counter is enabled for counting. When set to one the counter is enabled.	0	RW

18.5.6 Watchdog Timer

Register Name:	ost_wdog																															
Physical Address:	0xC8005014																Reset Hex Value:								0xFFFFFFFF							
Register Description:	Watchdog Counter																															
Access: Read/Write																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
tim_val																																

Register		ost_wdog		
Bits	Name	Description	Reset Value	Access
31:0	tim_val	Timer value. *Write has no effect unless ost_wdog_key=0x482E	0xffffffff	RW



18.5.7 Watchdog Enable Register

Register Name:		ost_wdog_enab																													
Physical Address:		0xC8005018														Reset Hex Value:							0x00000000								
Register Description:		Watchdog Enable Register																													
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																										cnt_en		int_en		rst_en	

Register		ost_wdog_enab		
Bits	Name	Description	Reset Value	Access
31:3	(Reserved)		29'd0	RO
2	wdog_cnt_ena	Count Enable. '1' enables ost_wdog to decrement. * Write has no effect unless ost_wdog_key=0x482E	0	RW
1	wdog_int_ena	Interrupt Enable. '1' enables ost_wdog_int signal to be generated. *Only writable when ost_wdog_key=0x482E	0	RW
0	wdog_rst_ena	Watchdog Reset Enable. '1' enables ost_wdog_reset_n signal to be generated. *Write has no effect unless ost_wdog_key=0x482E	0	RW

18.5.8 Watchdog Key Register

Register Name:	ost_wdog_key																																
Physical Address:	0xC800501C																Reset Hex Value:								0x00000000								
Register Description:	Watchdog Key Register																																
Access: Read/Write																																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
(Reserved)																key_value																	

Register		ost_wdog_key		
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)		16'd0	RO
15:0	key_value	Reset Key Value. Value of the reset key in which access is allowed to ost_wdog_enab and ost_wdog, ost_wdog_key=0x482E	16'd0	RW



18.5.9 Timer Status

Register Name:	ost_status																															
Physical Address:	0xC8005020														Reset Hex Value:							0x00000000										
Register Description:	Timer Status Register																															
Access: Read/Write 1 to Clear																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(Reserved)																													ost status bits			

Register		ost_status		
Bits	Name	Description	Reset Value	Access
31:5	(Reserved)	Reserved. Returns 0 if read	27'd0	RO
4	warm_reset	'1' if warm reset has occurred. Writing a '1' to this bit will clear it if set.	0	RW1C
3	ost_wdog_int_val	'1' if ost_wdog_int has occurred. Writing a '1' to this bit will clear it if the condition that caused it is no longer present.	0	RW1C
2	ost_ts_int_val	'1' if ost_ts_int has occurred. Writing a '1' to this bit will clear it if the condition that caused it is no longer present.	0	RW1C
1	ost_tim1_int_val	'1' if ost_tim1_int has occurred. Writing a '1' to this bit will clear it if the condition that caused it is no longer present.	0	RW1C
0	ost_tim0_int_val	'1' if ost_tim0_int has occurred. Writing a '1' to this bit will clear it if the condition that caused it is no longer present.	0	RW1C

18.5.10 Timestamp Compare Register

Register Name:	ost_ts_cmp																															
Physical Address:	0xC8005024																Reset Hex Value:								0x00000000							
Register Description:	Timestamp Compare Register																															
Access: Read/Write																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
compare_val																																

Register		ost_ts_cmp		
Bits	Name	Description	Reset Value	Access
31:0	compare_val	Value of the compare register. When the time stamp register reaches the value of the compare value, an interrupt is set and will not be cleared until it is cleared by writing to the timer status register.	32'h0	RW



18.5.11 Timestamp Configuration Register

Register Name:				ost_ts_cfg																											
Physical Address:				0xC8005028				Reset Hex Value:				0x00000000																			
Register Description:				Timestamp Configuration Register																											
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																ts_pause_en		ts_scale_en													

Register		ost_ts_cfg			
Bits	Name	Description	Reset Value	Access	
31:2	(Reserved)		30'd0	RO	
1	ts_pause_en	If '0' the timestamp counter is running, when '1' the counter is paused. Only when the prescale/scale_en is active. This bit has no effect if the timer is not using the prescaler/scale_en	0	RW	
0	ts_scale_en	If this field is '1', the 20-ns scale is enabled. It will make timestamp timer count with 3/4 of the frequency the timestamp timer is driven by. If no prescaler is used apb_clk.	0	RW	

18.5.12 Timestamp Prescale Register

Register Name:	ost_ts_pre																																
Physical Address:	0xC800502C																Reset Hex Value:								0x00000000								
Register Description:	Timestamp Prescale Register																																
Access: Read/Write																																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
(Reserved)																ts_presc																	

Register		ost_ts_pre			
Bits	Name	Description	Reset Value	Access	
31:16	(Reserved)		16'h0000	RO	
15:0	ts_presc	Contains the divide factor for the Timestamp Timer. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor. The prescaling starts when the field is written.	16'h0000	RW	



18.5.13 General-Purpose Timer 0 Configuration Register

Register Name:		ost_tim0_cfg																													
Physical Address:		0xC8005030				Reset Hex Value:				0x00000000																					
Register Description:		General-Purpose Timer 0 Configuration																													
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																tim0_pause_en		tim0_scale_en		tim0_bit1		tim0_bit0									

Register		ost_tim0_cfg															
Bits	Name	Description														Reset Value	Access
31:4	(Reserved)															28'd0	RO
3	tim0_pause_en	If '0' the timer0 counter is running, when '1' the counter is stopped. Only when the prescale/scale_en is active. This bit has no effect if the timer is not using the prescaler/scale_en														0	RW
2	tim0_scale_en	If this field is '1', the 3/4 scale is enabled. It will make the tim0 count with 3/4 of the frequency the timer0 is driven by. If no prescaler is used, it will be 3/4 of the apb_clk.														0	RW
1	tim0_bit1	This is bit 1 in a higher granularity for the reload counter for tim0.														0	RW
0	tim0_bit0	This is bit 0 in a higher granularity for the reload counter for tim0.														0	RW

18.5.14 General-Purpose Timer 0 Prescale Register

Register Name:		ost_tim0_pre																													
Physical Address:		0xC8005034								Reset Hex Value:				0x00000000																	
Register Description:		General-Purpose Timer 0 Prescale Register																													
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																tim0_presc															

Register		ost_tim0_pre															
Bits	Name	Description														Reset Value	Access
31:16	(Reserved)															16'h0000	RO
15:0	tim0_presc	Contains the divide factor for the timer0. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor, which is N+1, so when tim0_presc = 0x01 it is divide by 2. The prescaling starts when the field is written with a value other than 0x00.														16'h0000	RW



18.5.15 General-Purpose Timer 1 Configuration Register

Register Name:		ost_tim1_cfg																													
Physical Address:		0xC8005038				Reset Hex Value:				0x00000000																					
Register Description:		General-Purpose Timer 1Configuration																													
Access: Read/Write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																tim1_pause_en		tim1_scale_en		tim1_bit1		tim1_bit0									

Register		ost_tim1_cfg			
Bits	Name	Description	Reset Value	Access	
31:4	(Reserved)		28'd0	Read	
3	tim1_pause_en	If '0' the timer1 counter is running, when '1' the counter is stopped. Only when the prescale/scale_en is active. This bit has no effect if the timer uses apb_clk.	0	RW	
2	tim1_scale_en	If this field is '1', the 3/4 scale is enabled. It will make timer1 count with 3/4 of the frequency the timer1 is driven by. If no prescaler is used, it will be 3/4 of the apb_clk.	0	RW	
1	tim1_bit1	This is bit 1 in a higher granularity for the reload counter for tim1.	0	RW	
0	tim1_bit1	This is bit 0 in a higher granularity for the reload counter for tim1.	0	RW	

18.5.16 General-Purpose Timer 1 Prescale Register

Register Name:	ost_tim1_pre																																
Physical Address:	0xC800503C																Reset Hex Value:	0x00000000															
Register Description:	General-Purpose Timer 1Prescale Register																																
Access: Read/Write																																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
(Reserved)																tim1_presc																	

Register		ost_tim1_pre			
Bits	Name	Description	Reset Value	Access	
31:16	(Reserved)		16'h0000	Read	
15:0	tim1_presc	Contains the divide factor for timer1. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor, which is N+1, so when tim0_presc = 0x01 it is divide by 2. The prescaling starts when the field is written with a value other than 0x00	16'h0000	RW	

§ §



19.0 Time Synchronization Hardware Assist (TSYNC)

19.1 Overview

In a distributed control system containing multiple clocks, individual clocks tend to drift apart. Some kind of correction mechanism is necessary to synchronize the individual clocks to maintain global time, which is accurate to some requisite clock resolution. The IEEE* 1588 standard defines a precision clock synchronization protocol for networked measurement and control systems, including several messages used to exchange timing information. The hardware assist logic required to achieve precision clock synchronization using the IEEE 1588 standard depends on the implementation.

This document describes the hardware assist logic developed to achieve time synchronization on Ethernet.

19.2 Block Diagram

The following diagram provides a programming model of the TSync block, showing registers and interconnections.

Figure 10-1: XMIT and RECV Message Time and Delay Message Time

The diagram illustrates the internal components and data flow for XMIT and RECV message time and delay message time. The components are organized into two main sections: XMIT (top) and RECV (bottom, enclosed in a dashed box).

XMIT Section:

- GPIO:** Connected to the Accumulator and the System Time Clock.
- Accumulator (32 bit):** Receives data from the GPIO and the System Time Clock. It is connected to the Addend via a $+=$ operation.
- Addend (32 bit):** Frequency Scaling Value. It is connected to the Accumulator via a $+=$ operation.
- System Time Clock (64 bit):** low word always read/written 1st. It is connected to the Accumulator and the Target Time via a $>=$ operation.
- Target Time (64 bit):** It is connected to the System Time Clock via a $>=$ operation.
- Auxiliary Slave Mode Snapshot (64 bit):** It is connected to the System Time Clock and the Auxiliary Master Mode Snapshot via a $>=$ operation.
- Auxiliary Master Mode Snapshot (64 bit):** It is connected to the Auxiliary Slave Mode Snapshot and the Ethernet XMIT Snapshot via a $>=$ operation.

RECV Section:

- Channel Control (32 bit):** It is connected to the Channel Event via a $>=$ operation.
- Channel Event (32 bit):** It is connected to the Channel Control via a $>=$ operation.
- Ethernet XMIT Snapshot (64 bit):** It is connected to the Auxiliary Master Mode Snapshot and the Ethernet RECV Snapshot via a $>=$ operation.
- Ethernet RECV Snapshot (64 bit):** It is connected to the Ethernet XMIT Snapshot and the Sequence ID via a $>=$ operation.
- Sequence ID (16 bit):** It is connected to the Ethernet RECV Snapshot and the RECV Message Source UUID via a $>=$ operation.
- RECV Message Source UUID (48 bit):** It is connected to the Sequence ID and the Ethernet RECV Snapshot via a $>=$ operation.

Other Components:

- APB Bus:** Connects the XMIT and RECV sections to the rest of the system.
- Interrupt to Xscale processor, if enabled in Control/Store:** It is connected to the Auxiliary Master Mode Snapshot and the Ethernet XMIT Snapshot.
- Sync/Delay Message Detect:** It is connected to the Ethernet interface and the Ethernet XMIT Snapshot.
- Ethernet:** It is connected to the Sync/Delay Message Detect and the Ethernet XMIT Snapshot.

Notes:

- If Master, the XMIT snapshot holds the "SYNC" msg time and the RECV snapshot holds the: "DELAY" msg time. If Slave, the XMIT snapshot holds the "DELAY" msg time and the RECV snapshot holds the "SYNC" msg time
- Take snapshot = Sync, edge detect, & lock until reset by write of '1' to corresponding bit in event register

Time synchronization adjusts the rate that System Time increases in a time slave so that it is synchronized to the time master's System Time. System Time is incremented by an overflow of the Accumulator. The Accumulator increments due to the repetitive addition of the Addend register to itself on every system clock. Therefore, periodically adjusting the value of the Addend register controls the rate that System time increments. System Time snapshots are taken in hardware by both master and slave when certain messages are detected. These snapshots are used to calculate the skew between master and slave, and the slave is then adjusted accordingly.

Per the 1588 specification, synchronized time is referenced to the end of the “start of frame delimiter” (SFD) as shown in [Figure 189](#). Therefore, the time sync hardware captures the system time immediately upon detection of the SFD in the appropriate snapshot register (two per channel, one for transmit, one for receive). Due to PHY and

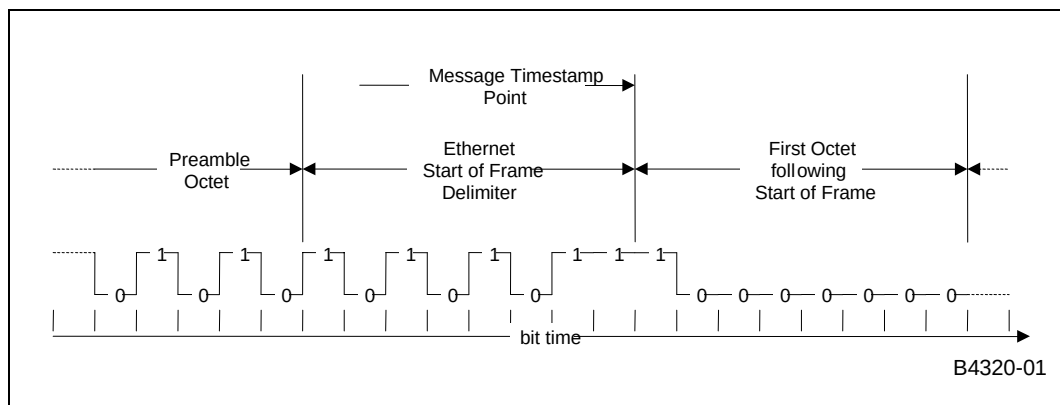


synchronization delays, the actual timestamp will be slightly later than the desired reference point. However, allowing for 1 pclk synchronization jitter, this is a fixed delay, easily nulled out in the software portion of the algorithm. This fixed delay is dependent on the 10/100 MHz selection at the PHY and therefore software needs to be able to access this information from each of the PHYs via the shared Management Data Interface (MDI).

When a Sync or Delay_Req messages is detected, the timestamp that was captured at the SFD in the snapshot register is frozen or “locked” until acknowledged by the firmware.

Each message is detected by identifying key bytes in the packet. The byte offsets identified below are numbered starting with the 1st byte after the SFD and the numbering begins at 0. Therefore “byte 74” refers to the 75th byte after the SFD.

Figure 189. Time Stamp Reference Point



19.3.1 Priority Message Support

Since there exists a wide variety of data on an industrial Ethernet network, including both time-critical and non-time-critical data, it is desirable to support “Tagged MAC Frames” from 802.3 which define priority based messages. A Tagged MAC Frames is identified by the “length/type” field. If byte 12 = 0x81 and byte 13=0x00, then the message is using the Tagged MAC Frame format, in which 4 additional bytes need to be accounted for in the header. Therefore, if a Tagged MAC Frame is detected, all the byte offsets mentioned below are incremented by 4.

19.3.2 Sync Message

Firmware in a time master transmits a multicast Sync message periodically over the network at 1, 2, 8, 16, or 64 second intervals. A Sync message is defined as a value of 0x00 in byte 74 of the Ethernet frame after the start of frame delimiter.

The TSync logic will monitor the MII signals, detect when the channel has transmitted or received a Sync message, and lock the timestamp. Furthermore, the TSync logic captures the Sequence ID and Source UUID if a Sync message is received by a channel configured as a time slave.

19.3.3 Follow_Up Message

Firmware in a time master transmits the timestamp, captured during a previously sent Sync message, using a multicast Follow_Up message. No time stamping is done by the master or slave with the Follow_Up message. A Follow_Up message is defined with a value of 0x02 in byte 74 of the Ethernet frame after the start of frame delimiter.



19.3.4 Delay_Req Message

Firmware in a time slave can transmit a Delay_Req message to the master for the purpose of determining propagation delays in the network. A Delay_Req message is defined as a value of 0x01 in byte 74 of the Ethernet frame after the start of frame delimiter.

The TSync logic will monitor the MII signals, detect when the channel has transmitted or received a Delay_Req message, and lock the timestamp. Furthermore, the TSync logic captures the Sequence ID and Source UUID if a Sync message is received by a channel configured as a time master.

19.3.5 Delay_Resp Message

Upon receipt of a Delay_Req message, firmware in a time master will transmit a Delay_Resp message that includes the timestamp of the Delay_Req message it received. No time stamping is done with the Delay_Resp message itself. A Delay_Resp message is defined with a value of 0x03 in byte 74 of the Ethernet frame after the start of frame delimiter.

The 1588 standard uses UDP/IP protocol, which implies that messages can be lost. Because a Delay_Req message can be locked out until firmware in the master channel enables it, the slave channel firmware will have to retry sending the Delay_Req message to the master if the slave does not receive a Delay_Resp message within some timeout period.

19.3.6 IPv6 Compatibility

A time sync message is always defined using an IPv4 message format. To avoid the potential of mistaking an IPv6 packet with a time sync packet, the byte at offset 14 will be checked for a value of 0x45. If not 0x45, the packet will be ignored. IPv6 is not supported.

19.3.7 Traffic Analyzer Support

In a traffic analyzer type application, it is often desirable to timestamp every message on the network. An optional mode inhibits the “locking” of the time snapshot so that the SFD of every message triggers a time snapshot. In this mode the snapshot must be read (presumably by the NPE) during the message before the next SFD is received.

19.3.8 MII Clocking Methods

According to the IEEE 802.3 specification, the MII TX/RX data transitions synchronously with the MII clock (rising edge is implied). For the TX case, txclk is driven by the PHY, but txdata is driven by MAC. The txdata bus transitions some time after rising edge of txclk and is sampled at next rising edge by the PHY. For the RX case, the PHY drives rxclk and rxdata. The rxdata bus transitions on the falling edge of rxclk to provide sufficient setup time for the MAC to sample at next rising edge. The characteristics of the MII in cases of transmit and receive should be taken into consideration when constraining the design for synthesis.

19.3.9 System Time Clock Rate Set by Addend Register

The FreqOscillator is the frequency of all portions of the frequency compensated clock, or time synchronization circuit. The frequency compensation value (FreqCompensationValue) is the number held in the Addend register, which is added to the accumulator once every 1/FreqOscillator. To determine the value that should be placed in the Addend register, the following equation is used:



$$\text{FreqDivisionRatio} = \text{FreqOscillator} / \text{FreqClock}$$

where FreqClock is the nominal frequency at which the clock counter is to be incremented.

The equation for the FreqCompensationValue utilizes the precision of the accumulator and the FreqDivisionRatio. Since the accumulator is 32 bits, the following equation applies:

$$\text{FreqCompensationValue} = 2^{32} / \text{FreqDivisionRatio}$$

The hexadecimal representation of the FreqCompensationValue is the value that is written to the Addend register. The following table gives examples of addend values based on a 66 MHz FreqOscillator.

Table 266. System Time Clock Rates

FreqOscillator	FreqClock	FreqDivisionRatio	FreqCompensationValue
66 MHz	40 MHz	1.65	0x9B26C9B3
66 MHz	50 MHz	1.32	0xC1F07C1F
66 MHz	60 MHz	1.1	0xE8BA2E8C

19.3.10 MII Message Detection

Two state machines for each Ethernet channel are implemented in the Time Sync logic: one of the pair is for transmit message detection and the other is for received message detection. Both are required for full duplex operation of the channel.

Mastership of a channel is indicated per channel in the TS_Channel_Control register. Based on this information, the Time Sync logic knows the mastership of the channel, and it monitors the MII signals for transmission or reception of Ethernet messages accordingly. A master channel will time stamp Sync messages that are transmitted and Delay_Req messages that are received. Conversely, a slave channel will time stamp Delay_Req messages that are transmitted and Sync messages that are received.

The timestamp point is immediately after the SFD and this value is frozen in the snapshot register when the last nibble of the frame CRC is transmitted or received and the overall message is detected. Because the Sync and Delay_Req messages are of fixed length, the location of the last nibble of CRC is known. Byte 169 corresponds to the last byte of the CRC. The snapshot is locked and this value is frozen until the software acknowledges it. Therefore, a constant can be subtracted from the snapshot to compensate for PHY and synchronization delays to arrive at the IEEE-1588 specified time stamp point.

An explanation of time stamping of messages and time stamp lockout is necessary to assist the user with the implementation of the TSync registers. Time stamping means that the current value in the system time register is captured in a second register, generally called a snapshot register. Each Ethernet channel has two snapshot registers, and there are two auxiliary snapshot registers controlled by general-purpose I/O.

The time stamping will occur when the appropriate conditions exist such as a general-purpose input is received or when a particular type of message is transmitted or received by the channel. Once a timestamp of system time is taken and locked, a unique indication for the snapshot is set in the appropriate Event register. No interrupt is sent to the Intel XScale® Processor upon timestamp capture/lock on the MII interface, this is due to being too early as the MII messages would not have propagated up the network protocol stack, thus the polling of the event register is necessary to determine if the timestamp is captured. No further timestamps of that type can be received until the snapshot indication is cleared by firmware. Thus, the setting of the indication is a lockout of further snapshots of a particular type until firmware takes action (unless the traffic analyzer lock inhibit feature is enabled).



Below is a description of the messages and how they are detected and utilized in the TSync block. For simplification, the description will reference a single channel and leave off the numeric descriptors as defined in the logic and register definitions.

19.3.10.1 Sync Message

The master channel multicasts a Sync message periodically over the network at 1-, 2-, 8-, 16-, or 64-second intervals. A Sync message is defined as a value of 0x00 in byte 74 of the Ethernet frame after the start of frame delimiter.

If the channel is a master, the Time Sync logic will monitor the MII signals and detect when the master channel has transmitted a Sync message. When the SFD is detected and the XMIT_Snapshot is not locked out, the message is time stamped and the current system time is captured in the XMIT_Snapshot register. If the channel is a slave, Sync messages will be received (not transmitted). When the SFD is detected and the RECV_Snapshot is not locked out, the message is time stamped and the current system time is captured in the RECV_Snapshot register. When the snapshot of system time related to the Sync message has occurred, an indication asserts in the TS_Channel_Event register and remains set until firmware explicitly writes a '1' back to that bit. Until the Sync message snapshot indication is cleared, no further Sync messages will be time stamped. Locking can be inhibited by setting the TS_Channel_Control register appropriately.

19.3.10.2 Delay_Req Message

Slave channels transmit a Delay_Req message to the master in response to receiving a Sync message. A Delay_Req message is defined as a value of 0x01 in byte 74 of the Ethernet frame after the start of frame delimiter.

If the channel is a master, the Time Sync logic will monitor the MII signals and detect when the master channel has received a Delay_Req message. When the SFD is detected and the RECV_Snapshot is not locked out, the message is time stamped and the current system time is captured in the RECV_Snapshot register. If the channel is a slave, Delay_Req messages will be transmitted (not received). When the SFD is detected and the XMIT_Snapshot is not locked out, the message is time stamped and the current system time is captured in the XMIT_Snapshot register. When the snapshot of system time related to the Delay_Req message has occurred, an indication asserts in the TS_Channel_Event register and remains set until firmware explicitly writes a '1' back to that bit. Until the Delay_Req message snapshot indication is cleared, no further Delay_Req messages will be time stamped. This is important to note since multiple slave channels may try to send Delay_Req messages simultaneously. Locking can be inhibited by setting the TS_Channel_Control register appropriately.

19.3.10.3 Errors in Messages

The TSync logic will ignore 1588 time-sync messages that are not properly formatted as described above. CRC errors or other errors detected by the MAC or higher levels of the protocol will cause firmware to delete the message and any time snapshots incorrectly captured by the TSync logic to be ignored.

Note: On the IXP45X/IXP46X network processors, the TSync logic does not check if properly formatted messages have PHY errors.



19.4 Theory of Operation (Auxiliary Snapshots)

19.4.1 Master Mode Programming Considerations

In master mode, the host processor firmware has complete control over both the incoming auxiliary snapshot signal and the clearing of the snapshot lock. Firmware will assert the signal, wait for the snapshot lock to set, read the snapshot, negate the signal, and then clear the lock. Since it has control over the environment, firmware will not clear the lock before it negates the signal. If the firmware does not clear the incoming auxiliary snapshot signal before the TSYNC lock is cleared, then a second / redundant snapshot event will be generated.

19.4.2 Slave Mode Programming Considerations

In slave mode, the host processor firmware typically knows the parameters of the signal coming from the master. The pulse per second signal from the GPS, for example, has a documented width, and the firmware must be designed to wait that amount of time after it detects the snapshot lock to the time it clears the lock. The firmware "wait" is facilitated by the fact that GPIO[8:7] should be configured as inputs to read this signal and assure that it is de-asserted before clearing the lock. Furthermore, the GPIO input can be configured to interrupt the CPU on the falling edge of the auxiliary snapshot signal, again making it easy for the firmware to know when to clear the lock. Hardware filtering and edge-detection were considered but not implemented because the signal quality from the master could be bad enough to cause spurious locks. For example, cables to a GPS could be a kilometer or more in length and the type of cable could be a factor as well.

Note: The host processor firmware handles the filtering and **MUST** not clear the lock until after the master has negated the snapshot input.

19.5 Detailed Register Descriptions

Table 267. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

19.5.1 Register Map

The registers of the TSync registers reside within the memory map of the IXP45X/IXP46X network processors. [Table 268](#) presents the address offset for the TSync registers, the names and mnemonics of the registers, and their access capability. Subsequent sections describe the contents of these registers in greater detail. The mnemonic names match names used in RTL code.



Table 268. Register Summary Table (Sheet 1 of 2)

Address Offset paddr[11:0]	Function	Mnemonic	Access
000	Time Sync Control	TS_Control	RW
004	Time Sync Event	TS_Event	RW
008	Addend	TS_Addend	RW
00C	Accumulator	TS_Accum	RW
010	Time Sync Test	TS_Test	RW
014	Unused	-	-
018	RawSystemTime_Low	TS_RSysTime_Lo	RO
01C	RawSystemTime_High	TS_RSysTimeHi	RO
020	SystemTime_Low	TS_SysTimeLo	RW
024	SystemTime_High	TS_SysTimeHi	RW
028	TargetTime_Low	TS_TrgtLo	RW
02C	TargetTime_High	TS_TrgtHi	RW
030	AuxSlaveModeSnap_Low	TS_ASMSLo	RO
034	AuxSlaveModeSnap_High	TS_ASMSHi	RO
038	AuxMasterModeSnap_Low	TS_AMMSLo	RO
03C	AuxMasterModeSnap_High	TS_AMMSHi	RO
040	TS_Channel0_Control	TS_Ch0_Control	RW
044	TS_Channel0_Event	TS_Ch0_Event	RW
048	XMIT_Snapshot0_Low	TS_TxSnap0Lo	RO
04C	XMIT_Snapshot0_High	TS_TxSnap0Hi	RO
050	RECV_Snapshot0_Low	TS_RxSnap0Lo	RO
054	RECV_Snapshot0_High	TS_RxSnap0Hi	RO
058	SourceUUID0_Low	TS_SrcUUID0Lo	RO
05C	SequenceID0/SourceUUID0_High	TS_SrcUUID0Hi	RO
060	TS_Channel1_Control	TS_Ch1_Control	RW
064	TS_Channel1_Event	TS_Ch1_Event	RW
068	XMIT_Snapshot1_Low	TS_TxSnap1Lo	RO
06C	XMIT_Snapshot1_High	TS_TxSnap1Hi	RO
070	RECV_Snapshot1_Low	TS_RxSnap1Lo	RO
074	RECV_Snapshot1_High	TS_RxSnap1Hi	RO
078	SourceUUID1_Low	TS_SrcUUID1Lo	RO
07C	SequenceID1/SourceUUID1_High	TS_SrcUUID1Hi	RO
080	TS_Channel2_Control	TS_Ch2_Control	RW
084	TS_Channel2_Event	TS_Ch2_Event	RW
088	XMIT_Snapshot2_Low	TS_TxSnap2Lo	RO
08C	XMIT_Snapshot2_High	TS_TxSnap2Hi	RO
090	RECV_Snapshot2_Low	TS_RxSnap2Lo	RO
094	RECV_Snapshot2_High	TS_RxSnap2Hi	RO
098	SourceUUID2_Low	TS_SrcUUID2Lo	RO



Table 268. Register Summary Table (Sheet 2 of 2)

Address Offset paddr[11:0]	Function	Mnemonic	Access
09C	SequenceID2/SourceUUID2_High	TS_SrcUUID2Hi	RO
0A0-0BF	Reserved for Channel 3	-	-
0C0-0DF	Reserved for Channel 4	-	-
0E0-0FF	Reserved for Channel 5	-	-
100-11F	Reserved for Channel 6	-	-
120-13F	Reserved for Channel 7	-	-
140-FFF	Unused	-	-

19.5.2 Register Descriptions

Table 269. Register Summary

Block Address	Offset Address	Register Name	Description	Reset Value	Page Number
	0x000	TS_Control	Time Sync Control Register	x0000	839
	0x004	TS_Event	Time Sync Event Register	x0010	840
	0x008	TS_Addend	Time Sync Addend Register	0x0	840
	0x00C	TS_Accum	Time Sync Accumulator Register	0x0	841
	0x010	TS_Test	Time Sync Test Register	x000	842
	0x018	TS_RSysTimeLo	RawSystemTime_Low Register	0x0	843
	0x01C	TS_RSysTimeHi	RawSystemTime_High Register	0x0	843
	0x020	TS_SysTimeLo	SystemTime_Low Register	0x0	844
	0x024	TS_SysTimeHi	SystemTime_High Register	0x0	844
	0x028	TS_TrgtLo	TargetTime_Low Register	0x0	845
	0x028	TS_TrgtHi	TargetTime_High Register	0x0	845
	0x030	TS_ASMSLo	Auxiliary Slave Mode Snapshot Low Register	0x0	846
	0x034	TS_ASMSHi	Auxiliary Slave Mode Snapshot High Register	0x0	846
	0x038	TS_AMMSLo	Auxiliary Master Mode Snapshot Low Register	0x0	847
	0x03C	TS_AMMSHi	Auxiliary Master Mode Snapshot High Register	0x0	847
	0x040*	TS_Ch_Control	Time Synchronization Channel Control Register	x00	848
	0x044*	TS_Ch_Event	Time Synchronization Channel Event Register	x00	849
	0x048*	TS_TxSnapLo	Transmit Snapshot Low Register	0x0	850
	0x04C*	TS_TxSnapHi	Transmit Snapshot High Register	0x0	851
	0x050*	TS_RxSnapLo	Receive Snapshot Low Register	0x0	852
	0x054*	TS_RxSnapHi	Receive Snapshot High Register	0x0	853
	0x058*	TS_SrcUUID0Lo	Source UUID0 Low Register	0x0	854
	0x05C*	TS_SrcUUIDHi	Sequence Identifier/Source UUID0 High Register	0x0	855



19.5.2.1 Time Sync Control Register

Register Name:						TS_Control																											
Block Base Address:						RegBlockAddress						Offset Address						0x000						Reset Value						x0000			
Register Description:						Time Sync Control Register																		Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
(Reserved)																								amm	asm	ttm	rst						

Register		TS_Control		
Bits	Name	Description	Reset Value	Access
31:4	(Reserved)	Reserved for future use.	x	x
3	amm	AMMS Interrupt Mask. Controls whether the Auxiliary Master Mode snapshot indication, which is the snm bit in the Time Sync Event register, should interrupt the Host processor. - When this bit is set, the interrupt to the Host is enabled. - When cleared, the AMMS interrupt to the Host is disabled.	0	RW
2	asm	ASMS Interrupt Mask. Controls whether the indication that an Auxiliary Slave Mode snapshot, which is the sns bit in the Time Sync Event register, has been taken should interrupt the Host processor. - When this bit is set, the interrupt to the Host is enabled. - When cleared, the ASMS interrupt to the Host is disabled.	0	RW
1	ttm	Target Time Interrupt Mask. Controls whether the Target Time interrupt is passed to the Host processor. - When this bit is set, the interrupt to the Host is enabled. - When cleared, the Target Time interrupt to the Host is disabled.	0	RW
0	rst	Reset. - When a '1' is written to this bit, all logic is returned to the same default state as when a power-on reset occurs. - After writing a '1' to this bit to reset the logic, the firmware must write a '0' to the bit to indicate the end of the reset.	0	RW



19.5.2.2 Time Sync Event Register

Register Name:						TS_Event																									
Block Base Address:						RegBlockAddress				Offset Address				0x004				Reset Value				x0010									
Register Description:						Time Sync Event Register																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																												amm	asm	ttm	rst

Register		TS_Event																Reset Value		Access	
Bits	Name	Description																			
31:4	(Reserved)	Reserved for future use.																x		x	
3	snm	AMMS Snapshot. This event bit sets when the system time register value is captured in the Auxiliary Master Mode Snapshot register upon an active high level on a general-purpose input, ammssig. - When this signal is asserted high, an interrupt will be generated to the Host on the ts_intreq if the amm bit in the Time Sync Control register is also set. - To clear snm, write a '1' to it.																0		RW	
2	sns	ASMS Snapshot. This event bit sets when the system time register value is captured in the Auxiliary Slave Mode Snapshot register upon detection of an active high level on a general-purpose input, asmssig. - When this signal is asserted high, an interrupt will be generated to the Host on the shared interrupt signal (ts_ntreq) if the asm bit in the Time Sync Control register is set. - To clear the sns bit, write a '1' to it.																0		RW	
1	ttipend	Target Time Interrupt Pending. This bit is the Target Time interrupt pending indication. When this bit is set, it indicates that the Target Time interrupt condition has occurred, which means that the System Time value has reached the 64-bit Target Time register value. - If ttm in the Time Sync Control register is set, the interrupt will be passed to the Host processor. - To clear this condition, the firmware must write a '1' to the ttipend bit. To prevent an immediate reoccurrence of the target time interrupt, the processor should first write a new value to the Target Time register and then clear the condition. This bit is set at power-up since both the System Time and the Target Time are reset at power-up to 0.																1		RW	
0	(Reserved)	Reserved for future use.																0		RW	

19.5.2.3 Addend Register

Register Name:						TS__Addend																									
Block Base Address:						RegBlockAddress				Offset Address				0x008						Reset Value				0x0							
Register Description:						Time Sync Addend Register																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Addend[31:0]																															



Register		TS_Addend		
Bits	Name	Description	Reset Value	Access
31:0	Addend	The Addend register contains the frequency scaling value used by a firmware algorithm to achieve time synchronization in the module. The value in this register is added to the value in the Accumulator. When the Accumulator rolls over, an overflow pulse is asserted and increments system time. Because the Addend register is cleared at reset, it must be written with a non-zero value to allow system time to increment.	0	RW

19.5.2.4 Accumulator Register

Register Name:						TS_Accum																									
Block Base Address:						RegBlockAddress					Offset Address					0x00C					Reset Value					0x0					
Register Description:						Time Sync Accumulator Register															Access:		(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Accumulator[31:0]																															

Register		TS_Accum		
Bits	Name	Description	Reset Value	Access
31:0	Accumulator	The Accumulator register serves as the frequency divider in the time synchronization logic. Firmware calculates a frequency scaling value to be written to the Addend register. The data in the Accumulator register is added to the value in the Addend register once every period of the system clock. When the Accumulator rolls over, an overflow pulse is asserted which increments the value in the system timer. This register is not read or written to in normal operation.	0	RW



19.5.2.5 Test Register

Register Name:						TS_Test																													
Block Base Address:						RegBlockAddress						Offset Address						0x010						Reset Value						x000					
Register Description:						Time Sync Test Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																												tenb		tm					

Register		TS_Test			
Bits	Name	Description	Reset Value	Access	
31:3	(Reserved)	Reserved for future use.	x	x	
2:1	tenb	Test Enable. These bits define what signals drive the ts_testmode_data pin when the tm bit in this register is set. The target time interrupt pending signal (readable in the TS_Event register) is driven if tenb[1:0] is '00' to support future applications. Specific system timer bits drive ts_testmode_data for the remaining settings of tenb[1:0]. tenb[1:0] ts_testmode_data TS_Event.ttipend TS_SysTimeLo[10] 10 TS_SysTimeLo[12] 11 TS_SysTimeLo[14]	0		
0	tm	Test Mode. This bit, which defaults to '0' at reset, is the test mode bit. When this bit is set, the TSync logic outputs one of four possible signals on the ts_testmode_data pin. The tenb[1:0] bits select the data. This data appears on a GPIO pin when the Test Mode bit is set.	0		



19.5.2.6 RawSystemTime_Low Register

Register Name:						TS_RSysTimeLo																													
Block Base Address:						RegBlockAddress						Offset Address						0x018						Reset Value						0x0					
Register Description:						RawSystemTime_Low Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
RawSystemTime_Low[31:0]																																			

Register		TS_RSysTimeLo															
Bits	Name	Description														Reset Value	Access
31:0	RawSystemTime_Low	This system time register is a read-only register of the raw system time. It is, therefore, not loadable and reflects the local time in the module. - The lower 32 bits of the 64-bit system time are read in this register. - The upper 32 bits are read in the RawSystemTime_High register. When a user reads system time with this pair of registers, no latching of system time occurs, which means that the system time could increment between the reading of the lower 32 bits in this register and the upper 32 bits in the RawSystemTime_High register. The user must account for this and deal with possible increments between reads of the two registers in firmware.														0	RO

19.5.2.7 RawSystemTime_High Register

Register Name:						TS_RSysTimeHi																													
Block Base Address:						RegBlockAddress						Offset Address						0x01C						Reset Value						0x0					
Register Description:						RawSystemTime_High Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
RawSystemTime_High[31:0]																																			

Register		TS_RSysTimeHi															
Bits	Name	Description														Reset Value	Access
31:0	RawSystemTime_High	This register contains the upper 32 bits of system time. When you want to read or write the system time, this register typically first accesses the RawSystemTime_Low Register. This register pair contains the raw system timer value, and no latching of system time occurs when the lower half is read. Time could increment between the reading of the lower 32 bits in the RawSystemTime_Low register and the reading of this register.														0	RO



19.5.2.8 SystemTime_Low Register

Register Name:					TS_SysTimeLo																										
Block Base Address:		RegBlockAddress					Offset Address					0x020					Reset Value					0x0									
Register Description:					SystemTime_Low Register															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SystemTime_Low[31:0]																															

Register		TS_SysTimeLo															
Bits	Name	Description														Reset Value	Access
31:0	SystemTime_Low	The system timer is a loadable up-counter, and reflects the local time in the module. While the system timer is 64 bits wide, the lower 32 bits reside in this register. The system timer is clocked by the module system clock and incremented when the Accumulator register rolls over. To read the entire system time value, the user must read this location first. Reading this location captures the upper 32 bits of the system time in a temporary register, which is accessed when the user reads the SystemTime_High Register next. Likewise, the SystemTime_Low Register must be written first when the user wants to write a new 64-bit value to system time. The data written to this register is captured in a holding register. When the user writes to the SystemTime_High Register, all 64 bits are then written to the system timer. Updating the system time with a direct write has precedence over increments to the system time based on an Accumulator rollover.														0	RW

19.5.2.9 SystemTime_High Register

Register Name:						TS_SysTimeHi																													
Block Base Address:						RegBlockAddress						Offset Address						0x024						Reset Value						0x0					
Register Description:						SystemTime_High Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
SystemTime_High[31:0]																																			

Register		TS_SysTimeHi															
Bits	Name	Description														Reset Value	Access
31:0	SystemTime_High	This register contains the upper 32 bits of system time. When the user wants to read or write the system time, this register must first access the SystemTime_Low Register. See Section 19.5.2.8, "SystemTime_Low Register" on page 844 for more details.														0	RW



19.5.2.10 TargetTime_Low Register

Register Name:						TS_TrgtLo																											
Block Base Address:						RegBlockAddress						Offset Address						0x028						Reset Value						0x0			
Register Description:						TargetTime_Low Register																		Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
TargetTime_Low[31:0]																																	

Register		TS_TrgtLo															
Bits	Name	Description														Reset Value	Access
31:0	TargetTime_Low	The Target Time register set contains 64 bits of a time value. When the system time is greater than or equal to the target time value, an interrupt is generated to the Host on the ts_intreq signal if the ttm bit in the Time Sync Control register is set. For more information about the Target Time interrupt, see Section 19.5.2.1, "Time Sync Control Register" on page 839.														0	RW

19.5.2.11 TargetTime_High Register

Register Name:						TS_TrgtHi																													
Block Base Address:						RegBlockAddress						Offset Address						0x028						Reset Value						0x0					
Register Description:						TargetTime_High Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
TargetTime_Low[31:0]																																			

Register		TS_TrgtLo															
Bits	Name	Description														Reset Value	Access
31:0	TargetTime_High	The Target Time register set contains 64 bits of a time value. When the system time is greater than or equal to the target time value, an interrupt is generated to the Host on the ts_intreq signal if the ttm bit in the Time Sync Control register is set. For more information about the Target Time interrupt, see Section 19.5.2.1, "Time Sync Control Register" on page 839.														0	RW



19.5.2.12 Auxiliary Slave Mode Snapshot Low Register – ASMS_Low

Register Name:					TS_ASMSLo																										
Block Base Address:					RegBlockAddress					Offset Address					0x030						Reset Value						0x0				
Register Description:					Auxiliary Slave Mode Snapshot Low Register																		Access:				(See below.)				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASMS_Low[31:0]																															

Register		TS_ASMSLo		
Bits	Name	Description	Reset Value	Access
31:0	ASMS_Low	When the board is operating in Slave mode, an active high level on a general-purpose input, asmssig, triggers a snapshot of System Time into the ASMS_Low and ASMS_High registers. The general-purpose input is synchronized by the Time Sync logic before it is used. Note: The processor can configure the GPIO bit as an output, but it will always be input-only to the Time Sync block. When the ASMS snapshot occurs, the sns indication in the Time Sync Event register is set. Writing a logic 1 to that bit clears the snapshot indication and allows a new snapshot to occur on the next rising transition of asmssig.	0	RO

19.5.2.13 Auxiliary Slave Mode Snapshot High Register – ASMS_High

Register Name:					TS_ASMSHi																										
Block Base Address:					RegBlockAddress					Offset Address					0x034					Reset Value					0x0						
Register Description:					Auxiliary Slave Mode Snapshot High Register																		Access:			(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASMS_High[31:0]																															

Register		TS_ASMSHi			
Bits	Name	Description	Reset Value	Access	
31:0	ASMS_High	When the board is operating in Slave mode, an active high level on a general-purpose input, asmssig, triggers a snapshot of System Time into the ASMS_Low and ASMS_High registers. The general-purpose input is synchronized by the Time Sync logic before it is used. Note: The processor can configure the GPIO bit as an output, but it will always be input-only to the Time Sync block. When the ASMS snapshot occurs, the sns indication in the Time Sync Event register is set. Writing a logic 1 to that bit clears the snapshot indication and allows a new snapshot to occur on the next rising transition of asmssig.	0	RO	



19.5.2.14 Auxiliary Master Mode Snapshot Low Register – AMMS_Low

Register Name:						TS_AMMSLo																													
Block Base Address:						RegBlockAddress						Offset Address						0x038						Reset Value						0x0					
Register Description:						Auxiliary Master Mode Snapshot Low Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
AMMS_Low[31:0]																																			

Register		TS_AMMSLo																											
Bits	Name	Description																										Reset Value	Access
31:0	AMMS_Low	When the board is operating in Master mode, it receives a general-purpose input signal for synchronization of snapshots and time. This general-purpose input, ammssig, is synchronized by the system clock in the Time Sync logic before it is used. Note: The processor can configure the GPIO as an output, but it will always be an input-only to the Time Sync block. When the AMMS snapshot occurs, the snm indication in the Time Sync Event register is asserted. No new snapshots in the AMMS register pair are captured until the firmware writes a '1' back to the snm bit to clear the snapshot indication.																										0	RO

19.5.2.15 Auxiliary Master Mode Snapshot High Register – AMMS_High

Register Name:						TS_AMMSHi																													
Block Base Address:						RegBlockAddress						Offset Address						0x03C						Reset Value						0x0					
Register Description:						Auxiliary Master Mode Snapshot High Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
AMMS_High[31:0]																																			

Register		TS_AMMSHi																										Reset Value	Access
Bits	Name	Description																											
31:0	AMMS_High	When the board is operating in Master mode, it receives a general-purpose input signal for synchronization of snapshots and time. This general-purpose input, ammssig, is synchronized by the system clock in the Time Sync logic before it is used. Note: The processor can configure the GPIO as an output, but it will always be an input-only to the Time Sync block. When the AMMS snapshot occurs, the snm indication in the Time Sync Event register is asserted. No new snapshots in the AMMS register pair are captured until the firmware writes a '1' back to the snm bit to clear the snapshot indication.																										0	RO



19.5.2.16 TS_Channel_Control Register (Per Channel)

Register Name:						TS_Ch_Control																													
Block Base Address:						RegBlockAddress						Offset Address						0x040*						Reset Value						x00					
Register Description:						Time Synchronization Channel Control Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Reserved																														ta	mm				
*Address offsets per channel... Channel 0 = 0x040 Channel 1 = 0x060 Channel 2 = 0x080																																			

Register		TS_Ch_Control			
Bits	Name	Description		Reset Value	Access
31:2	(Reserved)	Reserved for future use.		x	x
1	ta	Timestamp All messages. - When this bit is set, the locking of the time snapshot registers is inhibited. Each message is timestamped at the reception of a start of frame delimiter (SFD), regardless of whether the message is a Sync or Delay Request message. The timestamp is captured by the Snapshot register which is never locked and therefore must be read before the next SFD is received. - When this bit is cleared, the timestamp taken after the SFD is frozen or locked when a valid Sync or Delay Request message is detected, until the software resets it.		0	RW
0	mm	Master Mode. - When this bit is set, it indicates that this channel is a time master on the network. - When cleared, this bit indicates that this channel is in slave mode. The default after reset is slave mode.		0	RW



19.5.2.17 TS_Channel_Event Register (Per Channel)

Register Name:						TS_Ch_Event																													
Block Base Address:						RegBlockAddress						Offset Address						0x044*						Reset Value						x00					
Register Description:						Time Synchronization Channel Event Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Reserved																														rxs	txs				
*Address offsets per channel... Channel 0 = 0x044 Channel 1 = 0x064 Channel 2 = 0x084																																			

Register		TS_Ch_Event			
Bits	Name	Description	Reset Value	Access	
31:2	Reserved	Reserved for future use.	x	x	
1	rxs	Receive Snapshot Locked. This bit is automatically set when a Delay_Req message in Master mode, or a Sync message in Slave mode, is received and the ta bit in the corresponding TS_Channel_Control register is clear. It indicates that the current system time value has been captured in the RECV_Snapshot register and that further changes to the RECV_Snapshot are now locked out. To clear this bit, write a '1' to it.	0	RW	
0	txs	Transmit Snapshot Locked. This bit is automatically set when a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted and the ta bit in the corresponding TS_Channel_Control register is clear. It indicates that the current system time value has been captured in the XMIT_Snapshot register and that further changes to the XMIT_Snapshot are now locked out. To clear this bit, write a '1' to it.	0	RW	



19.5.2.18 XMIT_Snapshot_Low Register (Per Channel)

Register Name:						TS_TxSnapLo																													
Block Base Address:						RegBlockAddress						Offset Address						0x048*						Reset Value						0x0					
Register Description:						Transmit Snapshot Low Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
XMIT_Snapshot_Low[31:0]																																			
*Address offsets per channel... Channel 0 = 0x048 Channel 1 = 0x068 Channel 2 = 0x088																																			

Register		TS_TxSnapLo															
Bits	Name	Description										Reset Value				Access	
31:0	XMIT_Snapshot_Low	When a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted, the current system time is captured in this XMIT_Snapshot register. - The XMIT_Snapshot_Low register contains the lower 32 bits of the time value. - The XMIT_Snapshot_High register contains the upper 32 bits. After a XMIT_Snapshot has occurred, the txs indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the XMIT_Snapshot_Low and XMIT_Snapshot_High registers before it writes a '1' to the txs bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.										0				RO	



19.5.2.19 XMIT_Snapshot_High Register (Per Channel)

Register Name:						TS_TxSnapHi																													
Block Base Address:						RegBlockAddress						Offset Address						0x04C*						Reset Value						0x0					
Register Description:						Transmit Snapshot High Register																		Access:			(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
XMIT_Snapshot_High[31:0]																																			
*Address offsets per channel... Channel 0 = 0x04C Channel 1 = 0x06C Channel 2 = 0x08C																																			

Register		TS_TxSnapHi		
Bits	Name	Description	Reset Value	Access
31:0	XMIT_Snapshot_High	When a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted, the current system time is captured in this XMIT_Snapshot register. • The XMIT_Snapshot_Low register contains the lower 32 bits of the time value. • The XMIT_Snapshot_High register contains the upper 32 bits. After a XMIT_Snapshot has occurred, the txs indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the XMIT_Snapshot_Low and XMIT_Snapshot_High registers before it writes a '1' to the txs bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO



19.5.2.20 RECV_Snapshot Low Register (Per Channel)

Register Name:					TS_RxSnapLo																											
Block Base Address:					RegBlockAddress					Offset Address					0x050*					Reset Value					0x0							
Register Description:					Receive Snapshot Low Register																		Access:					(See below.)				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RECV_Snapshot_Low[31:0]																																
*Address offsets per channel... Channel 0 = 0x050 Channel 1 = 0x070 Channel 2 = 0x090																																

Register		TS_RxSnapLo		
Bits	Name	Description	Reset Value	Access
31:0	RECV_Snapshot_Low	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the current system time is captured in this RECV_Snapshot register. • The RECV_Snapshot_Low register contains the lower 32 bits of the time value. • The RECV_Snapshot_High register contains the upper 32 bits. After a RECV_Snapshot has occurred, the rxs indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the RECV_Snapshot_Low and RECV_Snapshot_High registers before it writes a '1' to the rxs bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO



19.5.2.21 RECV_Snapshot High Register (Per Channel)

Register Name:					TS_RxSnapHi																										
Block Base Address:					RegBlockAddress					Offset Address					0x054*					Reset Value					0x0						
Register Description:					Receive Snapshot High Register															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RECV_Snapshot_High[31:0]																															
*Address offsets per channel... Channel 0 = 0x054 Channel 1 = 0x074 Channel 2 = 0x094																															

Register		TS_RxSnapHi		
Bits	Name	Description	Reset Value	Access
31:0	RECV_Snapshot_High	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the current system time is captured in this RECV_Snapshot register. • The RECV_Snapshot_Low register contains the lower 32 bits of the time value. • The RECV_Snapshot_High register contains the upper 32 bits. After a RECV_Snapshot has occurred, the rxs indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the RECV_Snapshot_Low and RECV_Snapshot_High registers before it writes a '1' to the rxs bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO



19.5.2.22 SourceUUID0_Low Register (Per Channel)

Register Name:						TS_SrcUUID0Lo																													
Block Base Address:						RegBlockAddress						Offset Address						0x058*						Reset Value						0x0					
Register Description:						Source UUID0 Low Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
SourceUUID0_Low[31:0]																																			
*Address offsets per channel... Channel 0 = 0x058 Channel 1 = 0x078 Channel 2 = 0x098																																			

Register		TS_SrcUUID0Lo			
Bits	Name	Description	Reset Value	Access	
31:0	SourceUUID0_Low	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the Source UUID of the message is captured. The source UUID is located in bytes 64 through 69 of the Ethernet message, and this register contains the lower 32 bits of the source UUID. This register is read-only. At reset, the value in the register is 0, which is not a valid Source UUID value.	0	RO	



19.5.2.23 SequenceID / SourceUUID_High Register (Per Channel)

When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the source UUID and the sequence ID of the message are captured.

Register Name:					TS_SrcUUIDHi																										
Block Base Address:		RegBlockAddress					Offset Address					0x05C*					Reset Value					0x0									
Register Description:					Sequence Identifier/Source UUID0 High Register															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SequenceID[15:0]															SourceUUID_High[47:32]																
*Address offsets per channel...																															
Channel 0 = 0x05C																															
Channel 1 = 0x07C																															
Channel 2 = 0x09C																															

Register		TS_SrcUUIDHi		
Bits	Name	Description	Reset Value	Access
31:16	SequenceID	The sequence ID is located in bytes 72 and 73 of the Ethernet message, and is captured in this register in bit locations [31:16].	0	RO
15:0	SourceUUID_High	This register contains the upper 16 bits (bits 47:32) of the source UUID in bit locations [15:0].	0	RO

§ §





20.0 Synchronous Serial Port

The Synchronous Serial Port (SSP) is a full-duplex synchronous serial interface. It can connect to a variety of external analog-to-digital (A/D) converters, audio and telecom codecs, and many other devices that use serial protocols for transferring data. It supports National Microwire*, Texas Instruments* synchronous serial protocol (SSP), and Motorola* serial peripheral interface (SPI) protocol.

The SSP operates in master mode (the attached peripheral functions as a slave), and supports serial bit rates from 7.2 KHz to 1.84 MHz. Serial data formats may range from 4 to 16 bits in length. Two on-chip register blocks function as independent FIFOs for data, one for each direction. The buffers are 16 entries deep x 16 bits wide.

Buffers may be loaded or emptied by the system processor using SRAM-like transfers. The transfers are always one word per transfer. Each 32-bit word from the system fills one entry in a FIFO using the lower half 16-bits of a 32-bit word.

20.1 SSP Operation

Serial data is transferred between the system and an external peripheral through FIFO buffers in the SSP Port. Transfers are initiated by the host processor to/from system memory. Operation is full duplex — separate buffers and serial data paths permit simultaneous transfers in both directions.

20.1.1 Processor-Initiated Data Transfer

Transmit data (system to peripheral) is written by host processor to SSP Port "Transmit" buffer. The buffer works as a FIFO, and is seen as one 32-bit location by the processor. The SSP Port then takes the data from the buffer, serializes it, and sends it over the serial wire (**SSP_TXD**) to the peripheral.

Received data from peripheral (on **SSP_RXD**) converted to parallel words and stored in "Receive" FIFO buffer. A programmable "fullness" threshold, when passed, triggers an interrupt to the interrupt controller (hence, if enabled, to an interrupt input to the CPU). Interrupt service routine responds by identifying source of interrupt and then doing an SRAM-like read from inbound FIFO buffer. All buffer reads and writes use SPB to transfer data between internal CPU system bus and the buffer.

The host processor differentiates between the two FIFOs by whether it does a READ or a WRITE transfer. Reads automatically target the Receive FIFO, while writes will write data to the Transmit FIFO. From a memory map point of view, they are at the same address.

FIFO buffers are 16 words deep x 16 bits wide. This stores up to 16 samples per buffer.



20.2 Data Formats

20.2.1 Serial Data Formats for Transfer to/from Peripherals

Four pins are used to transfer data between the CPU and external codecs or modems. Although there are several formats for serial data, they have the same basic structure:

- **SSP_SCLK** — Defines the bit rate at which serial data is driven onto, and sampled from, the bus;
- **SSP_SFRM** — Defines the boundaries of a basic data “unit”, comprised of multiple serial bits;
- **SSP_TXD** — Is the serial data path for outbound data, from system to peripheral;
- **SSP_RXD** — Is the serial data path for inbound data, from peripheral to system.

A data frame may contain from 4 to 16 bits, depending on the format selected. Serial data is transmitted MSB first. Three formats are supported: Motorola* SPI, Texas Instruments* synchronous serial protocol (SSP), and National Microwire*.

For SPI and Microwire* formats, **SSP_SFRM** functions as a chip select to enable the external device (target of the transfer), and is held active-low during the data transfer. For SSP format, **SSP_SFRM** is pulsed high for one (serial) data period at the start of each frame.

Function and use of the serial clock **SSP_SCLK** is different for each format:

- For Microwire, both data sources switch (change to the next bit) outgoing data on the falling edge of **SSP_SCLK**, and sample incoming data on the rising edge.
- For SSP, data sources switch outgoing data on the rising edge of **SSP_SCLK**, and sample incoming data on the falling edge
- SPI gives the user the choice of which edge of **SSP_SCLK** to use for switching outgoing data, and for sampling incoming data. In addition, the user can move the phase of **SSP_SCLK**, shifting its active state one-half period earlier or later at the start and end of a frame.

While SSP and SPI are full-duplex protocols, Microwire uses a half-duplex master-slave messaging protocol. At the start of a frame, a single-byte control message is transmitted from the controller to the peripheral; no data is sent by the peripheral. The peripheral interprets the message and, if it is a READ request, responds with requested data, one clock after the last bit of the requesting message. Return data - part of the same frame - can be from 4 to 16 bits in length. Total frame length is 13 to 25 bits.

Note that the serial clock (**SSP_SCLK**), if driven by the SSP Port, only toggles while active data transfer is underway. At other times it will be held in an inactive or idle state, as defined by the specified protocol under which it operates.

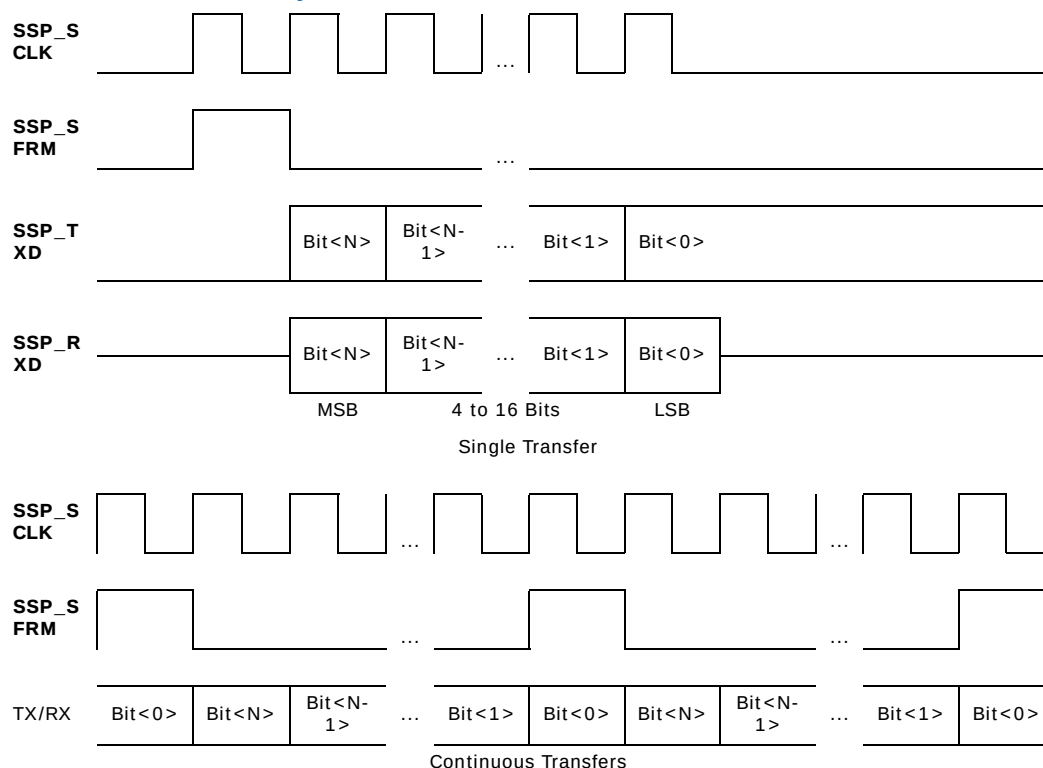


20.2.1.1 SSP Format — Detail

When outgoing data in the SSP controller is ready to transmit, **SSP_SFRM** asserts for one clock period. On the following clock, data to be transmitted is driven on **SSP_TXD** one bit at a time, most significant bit first. Similarly, the peripheral drives data on the **SSP_RXD** pin. Word length may be from 4 to 16 bits. All transitions take place on the rising edge of **SSP_SCLK** and data sampling is done on the falling edge. At the end of the transfer, **SSP_TXD** retains the value of the last bit sent (bit 0) through the next idle period. If the SSP Port is disabled or reset, **SSP_TXD** is forced to zero.

Table 270 shows the Texas Instruments synchronous serial frame format for a single transmitted frame and when back-to-back frames are transmitted. Once the bottom entry of the transmit FIFO contains data, **SSP_SFRM** is pulsed high for one **SSP_SCLK** period and the value to be transmitted is transferred from the transmit FIFO to the transmit logic's serial shift register. On the next rising edge of **SSP_SCLK**, the MSB of the 4 to 16-bit data frame is shifted to the **SSP_TXD** pin. Likewise, the MSB of the received data is shifted onto the **SSP_RXD** pin by the off-chip serial slave device. Both the SSP and the off-chip serial slave device then latch each data bit into their serial shifter on the falling edge of each **SSP_SCLK**. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of **SSP_SCLK** after the LSB has been latched.

Table 270. Texas Instruments* Synchronous Serial Frame Format



20.2.1.2 SPI Format — Detail

Note that there are four possible sub-modes of the SPI format depending on the **SSP_SCLK** edges selected for driving data and sampling received data, and on the selection of the phase mode of **SSP_SCLK** see "[SSP Control Register 1 \(SSCR1\)](#)" on [page 866](#) for complete description of each mode).



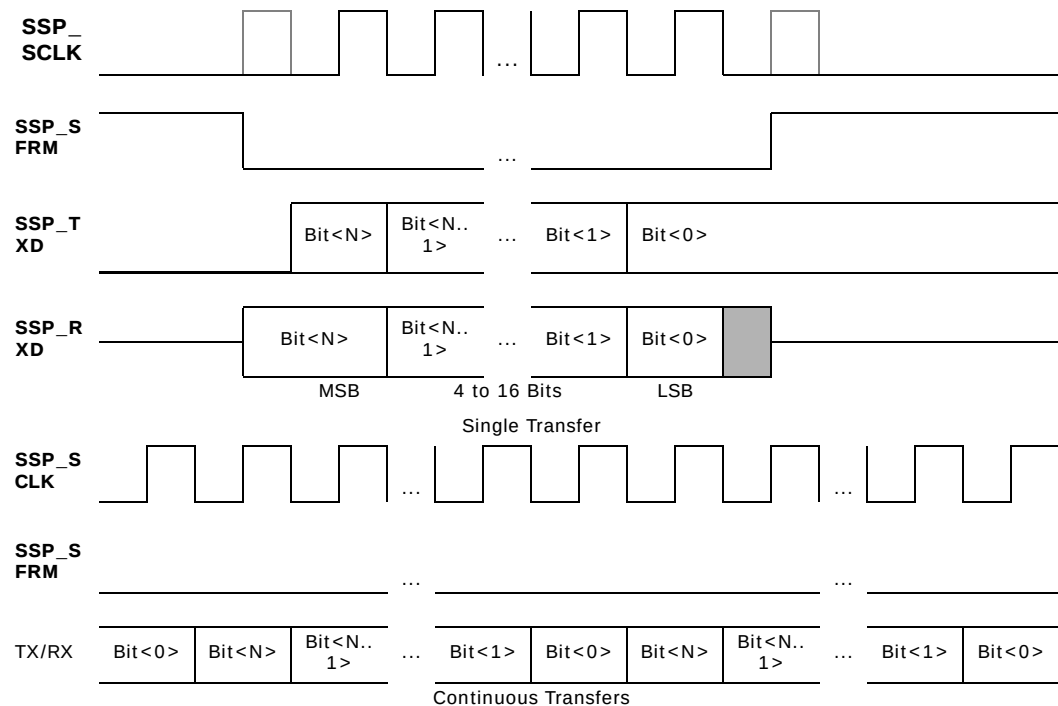
For SPI format, **SSP_SCLK** and **SSP_TXD** are low, and **SSP_SFRM** is high, in idle mode or when the SSP is disabled. When transmit (outgoing) data is ready, **SSP_SFRM** goes low and stays low for the remainder of the frame. The MSB of serial data is driven onto **SSP_TXD** a half-cycle later, and halfway into the first bit period **SSP_SCLK** asserts high and continues toggling for the remaining data bits. Data transitions on the falling edge of **SSP_SCLK**. From 4 to 16 bits may be transferred per frame.

With assertion of **SSP_SFRM**, receive data is simultaneously driven from the peripheral on **SSP_RXD**, MSB first. Data transitions on **SSP_SCLK** falling edges and is sampled by the controller on rising edges.

At the end of the frame, **SSP_SFRM** is de-asserted high one clock period after the last bit has been latched at its destination, and the completed incoming word is shifted into the “incoming” FIFO. The peripheral can tri-state **SSP_RXD** after sending the last (LSB) bit of the frame. **SSP_TXD** retains the last value transmitted when the controller goes into idle mode, unless the SSP port is disabled or reset (which forces **SSP_TXD** to zero).

For back-to-back transfers, start and completion are like those of a single transfer, but **SSP_SFRM** does not deassert between words. Both transmitter and receiver know the word length, and keep track internally of the start and end of words (frames). There are no “dead” bits; the least significant bit of one frame is followed immediately by the most significant bit of the next.

Table 271 shows one of the four possible configurations for the Motorola SPI frame format for a single transmitted frame and when back-to-back frames are transmitted.

**Table 271. Motorola* SPI Frame Format**

Note: The phase and polarity of **SSP_SCLK** can be configured for four different modes. This example shows just one of those modes.

20.2.1.3 Microwire* Format — Details

This format is similar to SPI, except transmission is half-duplex instead of full-duplex, using master-slave message passing.

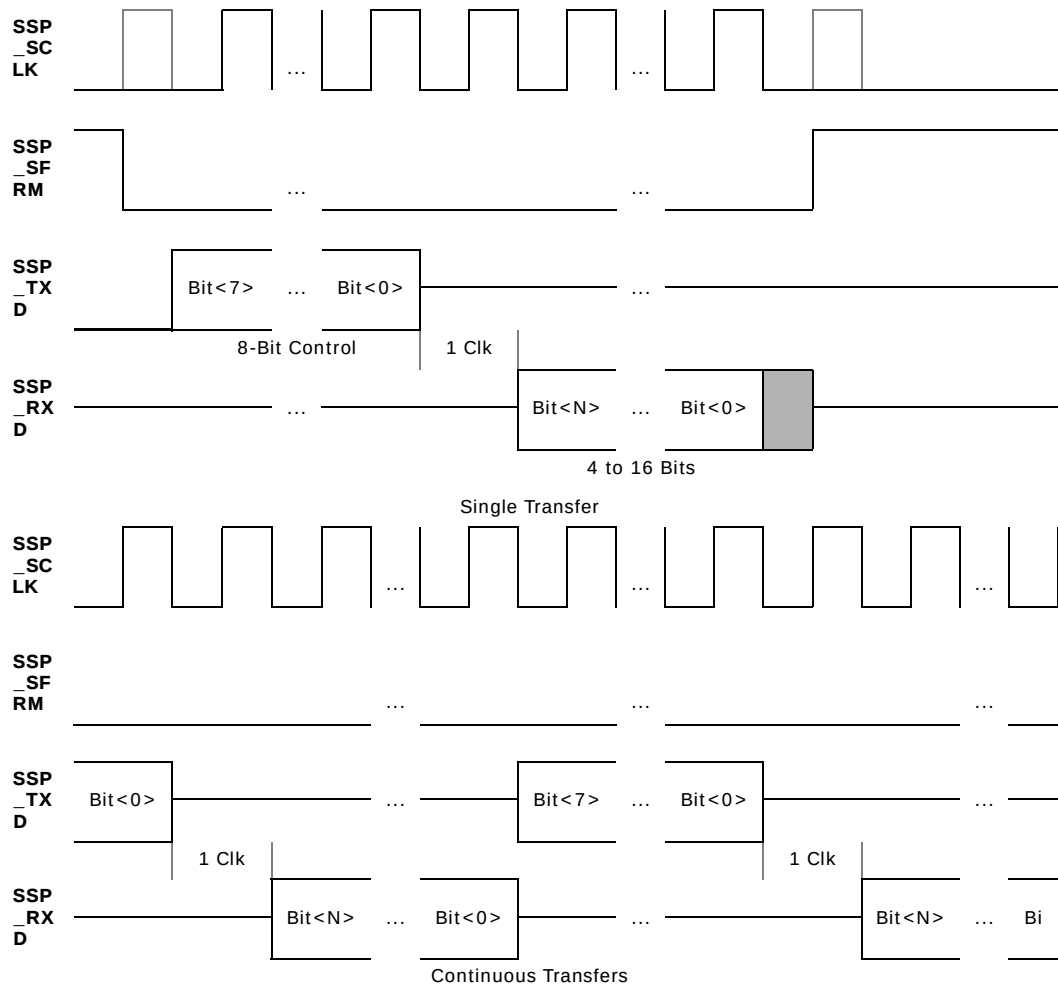
In idle state or when the SSP is disabled, **SSP_SCLK** is low, **SSP_SFRM** is high and **SSP_TXD** is low. Each serial transmission begins with **SSP_SFRM** assertion (low), followed by an 8-bit or 16-bit command word sent from controller to peripheral on **SSP_TXD**. **SSP_RXD**, controlled by the peripheral, remains tri-stated. **SSP_SCLK** goes high midway into the command MSB and continues toggling at the bit rate. One bit-period after the last command bit, the peripheral returns the serial data requested, MSB first, on **SSP_RXD**. Data transitions on falling edge of **SSP_SCLK** and is sampled on the rising edge. The last falling edge of **SSP_SCLK** coincides with the end of the last data bit (LSB) on **SSP_RXD**, and it remains low after that (if the only, or last word, of the transfer). **SSP_SFRM** deasserts high one-half clock period later.

The start and end of a series of back-to-back transfers are like those of a single transfer; however, **SSP_SFRM** remains asserted (low) throughout the transfer. The end of a data word on **SSP_RXD** is followed immediately by the start of the next command byte on **SSP_TXD**, with no “dead” time.

Table 272 shows the National Microwire frame format for a single transmitted frame and when back-to-back frames are transmitted.



Table 272. National Microwire* Frame Format



20.2.2 Parallel Data Formats for Buffer Storage

Data in buffers is always stored with one sample per 16-bit word regardless of the format data word length. Within each 16-bit field, stored data sample is right-justified, with word LSB in bit 0, and unused bits packed as zeroes on left-hand (MSB) side. Logic in SSP automatically left-justifies data in the Transmit FIFO so that the sample is properly transmitted on **SSP_TXD** in the selected frame format.

20.3 Buffer Operation

There are two separate and independent buffers for “incoming” (from peripheral) and “outgoing” (to peripheral) serial data. Buffers are filled or emptied by an SRAM-like transfer initiated by the system processor.

Although the system bus is 32 bits wide, only single samples may be transferred. Thus, only the lower two bytes of the 32 bit word will have valid data; the upper two bytes are not used and will include dummy or invalid data that should be discarded.



Each buffer consists of a dual-port register file with control circuitry to make it work as a FIFO, with independent read and write ports.

Buffer filling and emptying may be performed by the system processor in response to an interrupt from the FIFO logic. Each FIFO has a programmable threshold at which an interrupt is triggered. When the threshold value is exceeded, an interrupt is generated which, if enabled, signals the host processor to empty an “inbound” FIFO or to refill an “outbound” FIFO.

The system can also poll status bits to learn how full a FIFO is.

20.4 Baud-Rate Generation

The baud (or bit-rate clock) is generated internally by dividing the standard input clock (3.6864 MHz), which is first divided by 2. This feeds a programmable divider to generate baud rates from 7.2 KHz to 1.8432 Mbps. Optionally, an external clock (**SSP_EXTCLK**) can be used to replace the 3.6864 MHz standard input clock.

20.5 SSP Serial Port Registers

There are five registers in the SSP block: two control, one data, one status register, and one test register.

- Control registers are used to program the baud rate, data length, frame format, data transfer mechanism, and port enabling. In addition, they permit setting the FIFO “fullness” threshold that will trigger an interrupt.
- The Data Register is mapped as one 32-bit location, which physically points to either of two 32-bit registers. One register is for WRITES, and transfers data to the Transmit FIFO; the other is for READS, and takes data from the Receive FIFO. A write cycle will load successive words into the SSP Write Register, from the lower half 2 bytes of a 32-bit word to the Transmit FIFO. A READ cycle will similarly take data from the SSP Read Register, and the Receive FIFO will reload it with available data bits it has stored.

Read and writes should not increment the address; all accesses to the SSP Data Register memory address will access the Read or Write Register.

The FIFOs are independent buffers that allow full duplex operation.

- The Status Register signals the state of the FIFO buffers: whether the programmable threshold has been passed (Transmit/Receive Buffer service request), and a value showing the actual “fullness” of the FIFO. There are flag bits to indicate when the SSP is actively transmitting data, when the Transmit Buffer is not full, and when the Receive Buffer is not empty. Error bits signal overrun errors.

Table 273. SSP Serial Port Register Summary

Block Address	Register Name	Description	Reset Value	Page Number
0xC801_20	SSCR0	SSC Control Register 0	0x0000_0000	865
0xC801_20	SSCR1	SSC Control Register 1	0x0000_0000	869
0xC801_20	SSSR	SSP Status Register	0x0000_0000	871
0xC801_20	SSI TR	SSP Interrupt Test Register	0x0000_0000	872
0xC801_20	SSDR	SSP Data Register	0x0000_0000	873



Table 274. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

20.5.1 SSP Control Register 0 (SSCR0)

The SSP control register 0 (SSCR0) contains five different bit fields that control various functions within the SSP.

20.5.1.1 Data Size Select (DSS)

The 4-bit data size select (DSS) field is used to select the size of the data transmitted and received by the SSP. Data can be 4 to 16 bits in length. When data is programmed to be less than 16 bits, received data is automatically right-justified and the upper bits in the receive FIFO are zero-filled by receive logic. Transmit data should not be left-justified by the user before being placed in the transmit FIFO; transmit logic in the SSP will automatically left-justify the data sample according to the value of DSS before the sample is transmitted on **SSP_TXD**. Although it is possible to program data sizes of 1, 2, and 3 bits, these sizes are reserved and produce unpredictable results in the SSP.

When National Microwire frame format is selected, this bit field selects the size of the received data. Note that the size of the transmitted data is always 8 bits in this mode.

20.5.1.2 Frame Format (FRF)

The 2-bit frame format (FRF) field is used to select which frame format to use: Motorola SPI (FRF=00), Texas Instruments synchronous serial (FRF=01), or National Microwire (FRF=10). Note that FRF=11 is reserved and the SSP will produce unpredictable results if this value is used.

20.5.1.3 External Clock Select (ECS)

The external clock select (ECS) bit selects whether the on-chip 3.6864-MHz clock is used by the SSP or if an off-chip clock is supplied via **SSP_EXTCLK**. When ECS=0, the SSP uses the on-chip 3.6864-MHz clock to produce a range of serial transmission rates ranging from 7.2 Kbps to a maximum of 1.8432 Mbps. When ECS=1, the SSP uses **SSP_EXTCLK** to input a clock supplied from off-chip. The frequency of the off-chip clock can be any value up to 33.33 MHz. This off-chip clock is useful when a serial transmission rate, which is not an even multiple of 3.6864 MHz, is required for synchronization with the target off-chip slave device.

20.5.1.4 Synchronous Serial Port Enable (SSE)

The SSP enable (SSE) bit is used to enable and disable all SSP operations. When SSE=0, the SSP is disabled; when SSE=1, it is enabled. When the SSP is disabled, all of its clocks are powered down to minimize power consumption. Note that the SSE is within the SSP is reset to a known state. It is cleared to zero to ensure the SSP is disabled following a reset.



When the SSE bit is cleared during active operation, the SSP is disabled immediately, causing the current frame being transmitted to be terminated. Clearing SSE resets the SSP's FIFOs. However the SSP's control and status registers are not reset. The user must ensure these registers are properly reconfigured before re-enabling the SSP.

20.5.1.5 Serial Clock Rate (SCR)

The 8-bit serial clock rate (SCR) bit-field is used to select the baud, or bit rate, of the SSP. A total of 256 different bit rates can be selected, ranging from a minimum of 7.2 Kbps to a maximum of 1.8432 Mbps. The serial clock generator uses the 3.6864 MHz clock produced by the on-chip PLL divided by a fixed value of 2, and then divided by the programmable SCR value (0 to 255) plus 1 to generate the serial clock (**SSP_SCLK**). The resultant clock is driven on the **SSP_SCLK** pin and is used by the SSP's transmit logic to drive data on the **SSP_TXD** pin, and to latch data on the **SSP_RXD** pin. Depending on the frame format selected, each transmitted bit is driven on either the rising or falling edge of **SSP_SCLK**, and is sampled on the opposite clock edge.

The following bit table presents the bit locations corresponding to the five different control bit fields within SSP control register 0. Note that the SSE bit is the only control bit that is reset to a known state, to ensure the SSP is disabled following a reset. The reset state of all other control bits is unknown and must be initialized before enabling the SSP.

Register Name:						SSCR0																											
Block Base Address:						0xC801_20						Offset Address						0x00						Reset Value				0x0000_0000					
Register Description:						SSC Control Register 0														Access:				(See below.)									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved																SCR								SSE	ECS	FRF		SRS					

Register		SSCR0 (Sheet 1 of 2)		
Bit	Name	Description	Reset Value	Access
31:16	(Reserved)	(Reserved)	0x0000	RV
15:08	SCR	Serial Clock Rate Selection Value (0 to 255) used to generate transmission rate of SSP. Bit rate = $3.6864 \times 10^6 / (2 \times (SCR + 1))$ where SCR is a decimal integer	X	RW
7	SSE	Synchronous Serial Port Enable bit. 0 = SSP operation disabled 1 = SSP operation enabled	0x0000	RW
6	ECS	External clock select bit. 0 = On-chip clock used to produce the SSP's serial clock (SSP_SCLK). 1 = SSP_EXTCLK is used to create the SSP's SSP_SCLK .	0x0000	RW
5:4	FRF	This field specifies the Frame Format. 00 - Motorola* Serial Peripheral Interface (SPI) 01 - Texas Instruments* Synchronous Serial Protocol (SSP) 10 - National Microwire* 11 - Reserved, undefined operation	0x0000	RW



Register		SSCR0 (Sheet 2 of 2)		
Bit	Name	Description	Reset Value	Access
3:0	DSS	This field specifies the Data Size Selection. 0000 - Reserved, undefined operation 0001 - Reserved, undefined operation 0010 - Reserved, undefined operation 0011 - 4-bit data 0100 - 5-bit data 0101 - 6-bit data 0110 - 7-bit data 0111 - 8-bit data 1000 - 9-bit data 1001 - 10-bit data 1010 - 11-bit data 1011 - 12-bit data 1100 - 13-bit data 1101 - 14-bit data 1110 - 15-bit data 1111 - 16-bit data	0x0000	RW

20.5.2 SSP Control Register 1 (SSCR1)

The SSP Control Register 1 (SSCR1) contains nine bit fields that control various SSP functions.

20.5.2.1 Receive FIFO Interrupt Enable (RIE)

The Receive FIFO Interrupt Enable (RIE) bit is used to mask or enable the Receive FIFO service request interrupt. When RIE=0, the interrupt is masked, and the state of the Receive FIFO Service Request (RFS) bit within the SSP Status Register is ignored by the interrupt controller. When RIE=1, the interrupt is enabled, and whenever RFS is set to one an interrupt request is made to the interrupt controller. Note that programming RIE=0 does not affect the current state of RFS or the receive FIFO logic's ability to set and clear RFS, it only blocks the generation of the interrupt request.

20.5.2.2 Transmit FIFO Interrupt Enable (TIE)

The Transmit FIFO Interrupt Enable (TIE) bit is used to mask or enable the transmit FIFO service request interrupt. When TIE=0, the interrupt is masked and the state of the Transmit FIFO Service Request (TFS) bit within the SSP Status Register is ignored by the interrupt controller. When TIE=1, the interrupt is enabled, and whenever TFS is set to one an interrupt request is made to the interrupt controller. Note that programming TIE=0 does not affect the current state of TFS or the transmit FIFO logic's ability to set and clear TFS, it only blocks the generation of the interrupt request.

20.5.2.3 Loop Back Mode (LBM)

The loop back mode (LBM) bit is used to enable and disable the ability of the SSP transmit and receive logic to communicate. When LBM=0, the SSP operates normally. The transmit and receive data paths are independent and communicate via their respective pins. When LBM=1, the output of the transmit serial shifter is directly connected to the input of the receive serial shifter internally.



20.5.2.4 Serial Clock Polarity (SPO)

The serial clock (**SSP_SCLK**) polarity bit (SPO) selects the polarity of the inactive state of the **SSP_SCLK** pin when Motorola SPI format is selected (FRF=00). For SPO=0, the **SSP_SCLK** is held low in the inactive or idle state when the SSP is not transmitting/receiving data. For SPO=1, the **SSP_SCLK** is held high during the inactive/idle state. The programmed setting of the SPO alone does not determine which **SSP_SCLK** edge is used to transmit or receive data. The SPO setting in combination with the **SSP_SCLK** phase bit (SPH) determines this. Note that the SPO is ignored for all data frame formats except for the Motorola* SPI format (FRF=00).

20.5.2.5 Serial Clock Phase (SPH)

The serial clock (**SSP_SCLK**) phase bit (SPH) determines the phase relationship between the **SSP_SCLK** and the serial frame (**SSP_SFRM**) pins when the Motorola SPI format is selected (FRF=00). When SPH=0, **SSP_SCLK** remains in its inactive/idle state (as determined by the SPO setting) for one full cycle after **SSP_SFRM** is asserted low at the beginning of a frame. **SSP_SCLK** continues to transition for the rest of the frame and is then held in its inactive state for one-half of an **SSP_SCLK** period before **SSP_SFRM** is de-asserted high at the end of the frame. When SPH=1, **SSP_SCLK** remains in its inactive/idle state (as determined by the SPO setting) for one-half cycle after **SSP_SFRM** is asserted low at the beginning of a frame. **SSP_SCLK** continues to transition for the rest of the frame and is then held in its inactive state for one full **SSP_SCLK** period before **SSP_SFRM** is de-asserted high at the end of the frame. The combination of the SPO and SPH settings determines when **SSP_SCLK** is active during the assertion of **SSP_SFRM** and which **SSP_SCLK** edge is used to transmit and receive data on the **SSP_TXD** and **SSP_RXD** pins. When SPO and SPH are programmed to the same value (both 0 or both 1), transmit data is driven on the falling edge of **SSP_SCLK** and receive data is latched on the rising edge of **SSP_SCLK**. When SPO and SPH are programmed to opposite values (one 0 and the other 1), transmit data is driven on the rising edge of **SSP_SCLK** and receive data is latched on the falling edge of **SSP_SCLK**. Note that the SPH is ignored for all data frame formats except for the Motorola SPI format (FRF=00).

Table 275 shows the pin timing for all four programming combinations of SPO and SPH. Note that SPO inverts the polarity of the **SSP_SCLK** signal, and SPH determines the phase relationship between **SSP_SCLK** and **SSP_SFRM**, shifting the **SSP_SCLK** signal one-half phase to the left or right during the assertion of **SSP_SFRM**.

Table 275. Motorola* SPI Frame Formats for SPO and SPH Programming (Sheet 1 of 2)

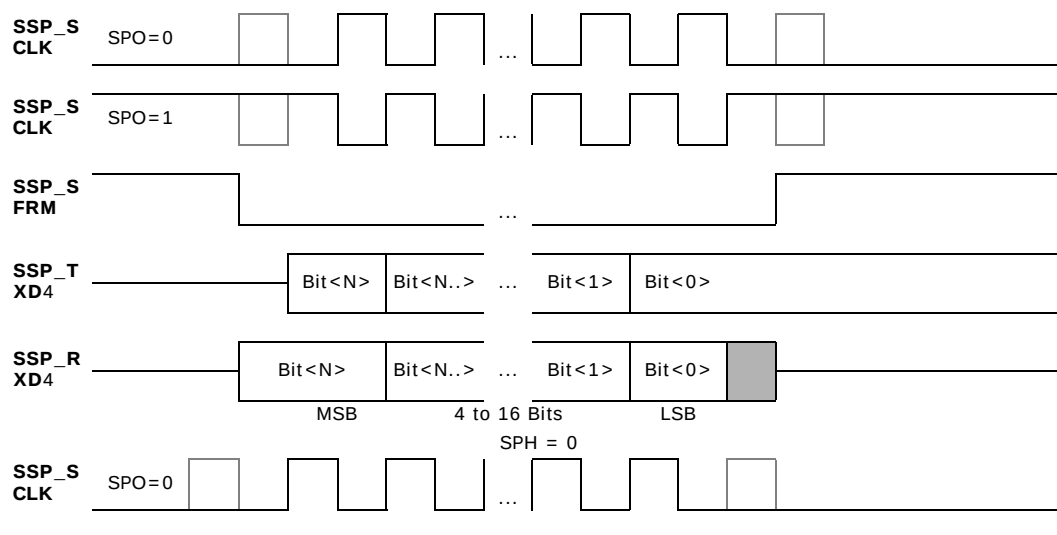
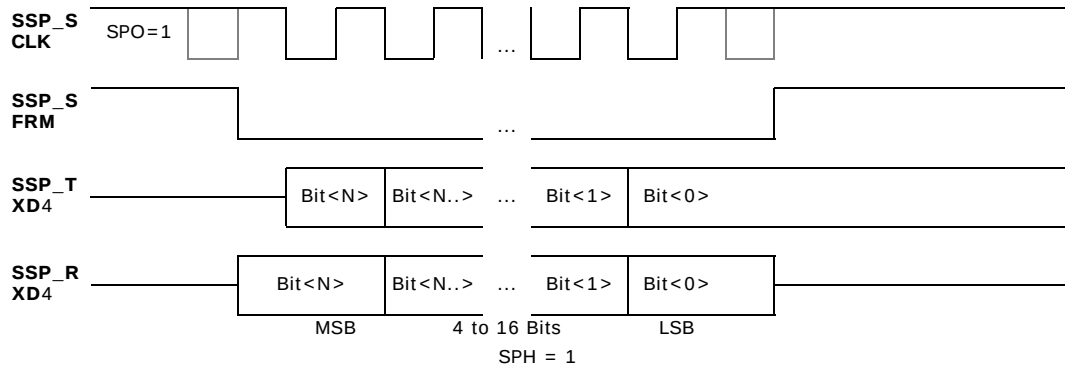




Table 275. Motorola* SPI Frame Formats for SPO and SPH Programming (Sheet 2 of 2)



20.5.2.6 National Microwire* Data Size (MWDS)

This bit sets the size of data in the Microwire format. If '1' a 16 bits data size is chosen for the Microwire format, otherwise, an 8-bit data size.

20.5.2.7 Transmit FIFO Interrupt Threshold (TFT)

This 4-bit value sets the level at or below which the FIFO controller triggers a service interrupt.

20.5.2.8 Receive FIFO Interrupt Threshold (RFT)

This 4-bit value sets the level at or above which the FIFO controller triggers a service interrupt.

20.5.2.9 Enable FIFO Write/Read Function (EFWR)

This bit enables a special functional mode for the SSP. When EFWR = 0, then the SSP operates in the normal mode described in this document. When EFWR = 1, then the SSP enters a mode in which whenever the CPU reads or writes to the SSP Data register it actually reads and writes exclusively to either the Transmit FIFO or the Receive FIFO depending on the programmed state of the Select FIFO for EFWR (STRF) bit. In this special mode, data will not be transmitted on the TXD pin and data input on the RXD pin will not be stored. This mode can be used to test, through software, whether or not the Transmit FIFO or the Receive FIFO operates properly as a first-in-first-out memory stack.

20.5.2.10 Select FIFO for Enable FIFO Write/Read (STRF)

This bit selects whether the Transmit or Receive FIFO is enabled for writes and reads whenever the EFWR is programmed to one, which puts the SSP in a special functional mode.

Note: The following bit table shows bit locations corresponding to control bit fields in SSP Control Register 1. Note that writes to reserved bits are ignored, and reads of these bits return zero.



Register Name:						SSCR1																													
Block Base Address:						0xC801_20						Offset Address						0x04						Reset Value						0x0000_0000					
Register Description:						SSC Control Register 1																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Reserved																STRF	EFWR	RFT				TFT				MWDS	SPH	SPO	LBM	TIE	RIE				

Register		SSCR1		
Bits	Name	Description	Reset value	Access
31:16	(Reserved)	(Reserved)	0x0000	RV
15	STRF	Select FIFO for EFWR: 0 = Transmit FIFO is selected for "enable FIFO write/read" 1 = Receive FIFO is selected for "enable FIFO write/read"	0x0000	RW
14	EFWR	Enable FIFO Write/Read: 0 = FIFO write/read loopback function is disabled; normal operation enabled 1 = FIFO write/read loopback function is enabled	0x0000	RW
13:10	RFT	Receive FIFO Threshold. Sets threshold level at which Receive FIFO asserts interrupt. This level should be set to the threshold value minus 1.	0x0000	RW
9:6	TFT	Transmit FIFO Threshold. Sets threshold level at which Transmit FIFO asserts interrupt. This level should be set to the threshold value minus 1.	0x0000	RW
5	MWDS	National Microwire* Data Size 0 = 8 bit Microwire format 1 = 16 bits Microwire format	0x0000	RW
4	SPH	Motorola* SPI SSP_SCLK phase setting: 0 = SSP_SCLK is inactive one full cycle at the start of a frame and 1/2 cycle at the end of a frame. 1 = SSP_SCLK is inactive 1/2 cycle at the start of a frame and one full cycle at the end of a frame.	0x0000	RW
3	SPO	Motorola SPI SSP_SCLK polarity setting: 0 = The inactive or idle state of SSP_SCLK is low. 1 = The inactive or idle state of SSP_SCLK is high.	0x0000	RW
2	LBM	Loop Bank Mode Enable bit. 0 = Normal serial port operation enabled 1 = Output of transmit serial shifter connected to input of receive serial shifter, internally	0x0000	RW
1	TIE	Transmit FIFO Interrupt Enable 0 = Transmit FIFO level interrupt is disabled 1 = Transmit FIFO level interrupt is enabled	0x0000	RW
0	RIE	Receive FIFO Interrupt Enable 0 = Receive FIFO level interrupt is disabled 1 = Receive FIFO level interrupt is enabled	0x0000	RW

20.5.3 SSP Status Register

The SSP status register (SSSR) contains bits that signal overrun errors as well as the transmit and receive FIFO service requests. Each of these hardware-detected events signal an interrupt request to the interrupt controller. The status register also contains flags that indicate when the SSP is actively transmitting characters, when the transmit FIFO is not full, and when the receive FIFO is not empty (no interrupt generated).



Bits that cause an interrupt will signal the request as long as the bit is set. Once the bit is cleared, the interrupt is cleared. Read/write bits are called status bits, read-only bits are called flags. Status bits are referred to as “sticky” (once set by hardware, must be cleared by software). Writing a one to a sticky status bit clears it, writing a zero has no effect. Read-only flags are set and cleared by hardware; writes have no effect. Additionally some bits that cause interrupts have corresponding mask bits in the control registers and are indicated in the section headings that follow.

20.5.3.1 Transmit FIFO Not Full Flag (TNF) (Read-Only, Non-Interruptible)

The transmit FIFO not full flag (TNF) is a read-only bit that is set whenever the transmit FIFO contains one or more entries that do not contain valid data. TNF is cleared when the FIFO is completely full. This bit can be polled when using programmed I/O to fill the transmit FIFO over its half-way mark. This bit does not request an interrupt.

20.5.3.2 Receive FIFO Not Empty Flag (RNE) (Read-Only, Non-Interruptible)

The receive FIFO not empty flag (RNE) is a read-only bit that is set whenever the receive FIFO contains one or more entries of valid data and is cleared when it no longer contains any valid data. This bit can be polled when using programmed I/O to remove remaining bytes of data from the receive FIFO since CPU interrupt requests are only made when the Receive FIFO Threshold has been met or exceeded. This bit does not request an interrupt.

20.5.3.3 SSP Busy Flag (BSY) (Read-Only, Non-Interruptible)

The SSP busy (BSY) flag is a read-only bit that is set when the SSP is actively transmitting and/or receiving data, and is cleared when the SSP is idle or disabled (SSE=0). This bit does not request an interrupt.

20.5.3.4 Transmit FIFO Service Request Flag (TFS) (Read-Only, Maskable Interrupt)

The Transmit FIFO service request flag (TFS) is a read-only bit that is set when the transmit FIFO is nearly empty and requires service to prevent an underrun. TFS is set any time the transmit FIFO has the same or fewer entries of valid data than indicated by the Transmit FIFO Threshold, and it is cleared when it has more entries of valid data than the threshold value. When the TFS bit is set, an interrupt request is made unless the transmit FIFO interrupt request enable (TIE) bit is cleared. After the CPU fills the FIFO such that it exceeds the threshold, the TFS flag (and the service request and/or interrupt) is automatically cleared.

20.5.3.5 Receive FIFO Service Request Flag (RFS) (Read-Only, Maskable Interrupt)

The receive FIFO service request flag (RFS) is a read-only bit that is set when the receive FIFO is nearly filled and requires service to prevent an overrun. RFS is set any time the receive FIFO has the same or more entries of valid data than indicated by the Receive FIFO Threshold, and it is cleared when it has fewer entries than the threshold value. When the RFS bit is set, an interrupt request is made unless the receive FIFO interrupt request enable (RIE) bit is cleared. After the CPU reads the FIFO such that it has fewer entries than the RFT value, the RFS flag (and the service request and/or interrupt) is automatically cleared.

20.5.3.6 Receiver Overrun Status (ROR)

The receiver overrun status bit (ROR) is a read/write bit that is set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. After the FIFO is filled, then each time a new piece of data is received, the set signal to



the ROR bit is asserted, and the newly received data is discarded. This process is repeated for each new piece of data received until at least one empty FIFO entry exists. When the ROR bit is set, an interrupt request is made. Writing 1 to this bit resets ROR status and its interrupt request.

After this ROR interrupt occurs, it is recommended that the user read the receive FIFO to empty it, then clear the ROR bit.

20.5.3.7 Transmit FIFO Level (TFL)

This 4-bit value shows how many valid entries are currently in the Transmit FIFO. See register table below for details.

20.5.3.8 Receive FIFO Level (RFL)

This 4-bit value shows how many valid entries are currently in the Receive FIFO. See register table below for details.

The following bit table shows the bit locations corresponding to the status and flag bits within the SSP status register. All bits are read-only except ROR, which is read/write. Writes to TNF, RNE, BSY, TFS, and RFS have no effect. Note that writes to reserved bits are ignored and reads to those bits return zeros.

Register Name:						SSSR																									
Block Base Address:						0xC801_20						Offset Address						0x08						Reset Value				0x0000_0000			
Register Description:						SSP Status Register																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																RFL				TFL				ROR	RFS	TFS	BSY	RNE	TNF	Rsvd.	

Register		SSSR (Sheet 1 of 2)		
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)	(Reserved)	0x0000	RV
15:12	RFL	Receive FIFO Level: Number of entries minus one in Receive FIFO. Note: When the value 0xF is read, the FIFO is either empty or full and the programmer should refer to the RNE bit.	0000	RO
11:08	TFL	Transmit FIFO Level: Number of entries in Transmit FIFO. Note: When the value 0x0 is read, the FIFO is either empty or full and the programmer should refer to the TNF bit.	0000	RO
7	ROR	Receive FIFO Overrun: 0 = Receive FIFO has not experienced an overrun 1 = Attempted data write to full Receive FIFO, request interrupt	0	RW
6	RFS	Receive FIFO Service Request: 0 = Receive FIFO level is below RFT threshold, or SSP disabled. 1 = Receive FIFO level is at or above RFL threshold, request interrupt	0	RO
5	TFS	Transmit FIFO Service Request: 0 - Transmit FIFO level exceeds TFT threshold, or SSP disabled 1 - Transmit FIFO level is at or below TFL threshold, request interrupt	0	RO



Register		SSSR (Sheet 2 of 2)		
Bits	Name	Description	Reset Value	Access
4	BSY	SSP is busy 0 = SSP is idle or disabled 1 = SSP currently transmitting or receiving a frame	0	RO
3	RNE	Receive FIFO not empty. 0 - Receive FIFO is empty 1 - Receive FIFO is not empty	0	RO
2	TNF	Transmit FIFO not Full. 0 = Transmit FIFO is full 1 = Transmit FIFO is not full	1	RO
1:0	(Reserved)	(Reserved)	00	RV

20.5.4 SSP Interrupt Test Register (SSITR)

Writing “1” to the corresponding bit position to the SSP Interrupt Test register generates an interrupt strobe signal to the Interrupt Controller or a DMA request for test purposes.

Note: SSITR functionality is available even when the SSP is disabled.

Register Name:						SSITR																									
Block Base Address:						0xC801_20				Offset Address				0x0C				Reset Value				0x0000_0000									
Register Description:						SSP Interrupt Test Register														Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																							TROR	TRFS	TTFS	Reserved					

Register		SSITR			
Bits	Name	Description	Reset Value	Access	
31:08	(Reserved)	(Reserved)	0x0000	RV	
7	TROR	Test Receive FIFO overrun (ROR)	0x0000	RW	
6	TRFS	Test Receive FIFO service request (RFS)	0x0000	RW	
5	TTFS	Test Transmit FIFO service request (TFS)	0x0000	RW	
4:0	(Reserved)	(Reserved)	0x0000	RV	

20.5.5 SSP Data Register (SSDR)

The SSP Data Register (SSDR) is a block of 32-bit locations that can be accessed by 32-bit data transfers. The SSDR represents two physical registers: the first is temporary storage for data on its way out through the Transmit FIFO, the other is temporary storage for data coming in through the Receive FIFO.

As the register is accessed by the system, FIFO control logic transfers data automatically between register and FIFO as fast as the system moves it. Data in the FIFO shifts up or down to accommodate the new word (unless it is an attempted WRITE to a full Transmit FIFO). Status bits are available to show the system whether either buffer is full, above/below a programmable threshold, or empty.



For outbound data transfers (WRITE from system to SSP peripheral), the register may be loaded (written) by the system processor anytime it is empty.

When a data size of less than 16-bits is selected, the user should not left-justify data written to the transmit FIFO. Transmit logic left-justifies the data and ignores any unused bits. Received data less than 16-bits is automatically right-justified in the receive buffer. When the SSP is programmed for National Microwire frame format, the default size for transmit data is 8-bits (the most significant byte is ignored), the receive data size is controlled by the programmer using the DSS field in SSCR0.

The following table shows the location of the SSP data register. Note that both FIFOs are cleared when the block is reset, or by writing a zero to SSE (SSP disabled).

Register Name:						SSDR																													
Block Base Address:						0xC801_20						Offset Address						0x10						Reset Value						0x0000_0000					
Register Description:						SSP Data Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Reserved																Data																			

Register		SSDR			
Bits	Name	Description	Reset Value	Access	
31:16	(Reserved)	(Reserved)	0x0000		
15:00	Data	Data (Low Word): Data word to be written to/read from the Transmit/Receive FIFO	0x0000		





21.0 I²C Bus Interface Unit

This chapter describes the I²C (Inter-Integrated Circuit) bus interface unit on the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors, including the operation modes and setup. Throughout this manual, this peripheral is referred to as the I²C unit.

21.1 Overview

The I²C Bus Interface Unit allows the IXP45X/IXP46X network processors to serve as a master and slave device residing on the I²C bus. The I²C bus is a serial bus developed by Phillips Corporation* consisting of a 2-pin interface. **SDA** (Serial Data/Address) is the data pin for input and output functions and **SCL** (Serial Clock Line) is the clock pin for reference and control of the I²C bus.

The I²C bus allows the IXP45X/IXP46X network processors to interface to other I²C peripherals and microcontrollers for system management functions. The serial bus requires a minimum of hardware for an economical system to relay status and reliability information on the IXP45X/IXP46X network processors to an external device.

The I²C Bus Interface Unit is a peripheral device that resides on the internal peripheral bus (APB) of the IXP45X/IXP46X network processors. Data is transmitted to and received from the I²C bus via a buffered interface. Control and status information is relayed through a set of memory-mapped registers. Refer to the I²C Bus Specification for complete details on I²C bus operation.

21.2 Feature List

The I²C is responsible for the following functions:

- Multi-Master capabilities.
- Slave capabilities.
- The I²C unit supports both fast-mode operation — at 400 Kbps — and standard mode — at 100 Kbps. Fast mode logic levels, formats, capacitive loading and protocols function the same in both modes. For further information, refer to *I²C Peripheral for Microcontrollers* by the Phillips Semiconductor.
- The I²C unit does *not* support I²C 10-bit addressing or CBUS.

21.3 Block Diagram

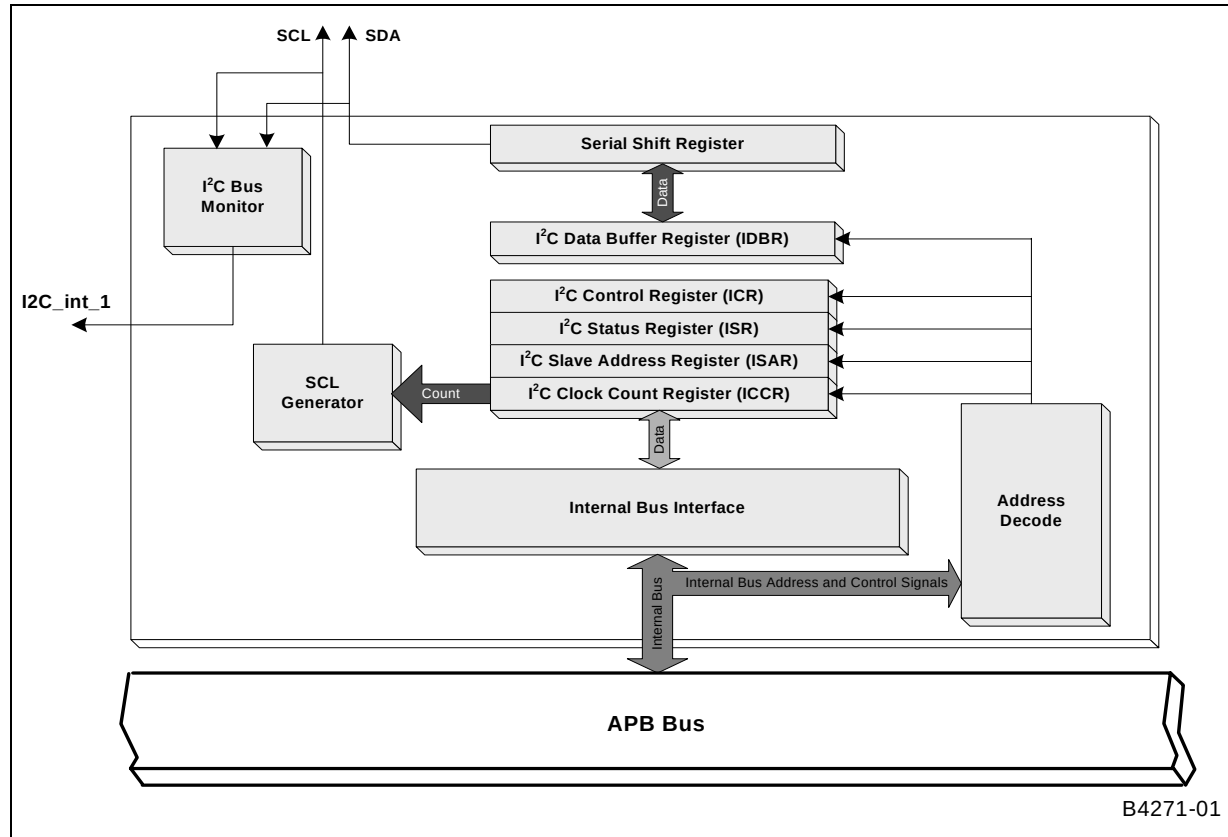
Figure 190 shows a block diagram of the I²C unit.

Fundamentally, the I²C bus consists of a single I/O data line (SDA), a single I/O clock, and synchronization line (SCL). These are the fundamental I²C signals. In the I²C unit of the IXP45X/IXP46X network processors, the SDA and SCL lines are controlled by logic and state machines which use a 33-MHz clock (clk_i2c) that is supplied by the CRU unit.



The I²C unit's state machines and functionality are partially controlled by using local memory mapped registers (MMRs) that store configuration and operation information. The registers are controlled from the APB bus and reside in the i2c_registers block.

Figure 190. I²C Bus Interface Unit Block Diagram



21.4 Theory of Operation

The I²C bus defines a serial protocol for passing information between agents on the I²C bus using a two pin interface. The interface consists of an **SDA** line and an **SCL**. Each device on the I²C bus is recognized by a unique 7-bit address and can operate as a transmitter or as a receiver. In addition to transmitter and receiver, the I²C bus uses the concept of master and slave.

Table 276 lists the I²C device types.

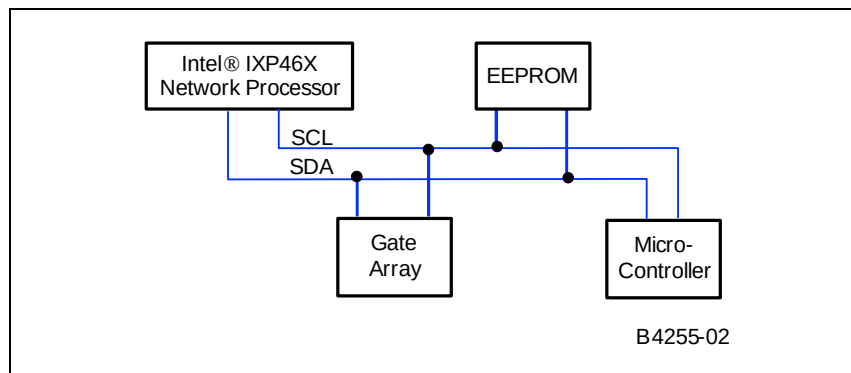
Table 276. I²C Bus Definitions (Sheet 1 of 2)

I ² C Device	Definition
Transmitter	Sends data to the I ² C bus.
Receiver	Receives data from the I ² C bus.
Master	Initiates a transfer, generates the clock signal, and terminates the transactions.

**Table 276. I²C Bus Definitions (Sheet 2 of 2)**

I ² C Device	Definition
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, when more than one master simultaneously tries to control the bus, only one is allowed. This procedure ensures that messages are not corrupted.

As an example of I²C bus operation, consider the case of the IXP45X/IXP46X network processors acting as a master on the bus (see [Figure 191](#)). The processor, as a master, addresses an EEPROM as a slave to receive data. The processor is a master-transmitter and the EEPROM is a slave-receiver. When the processor reads data, it is a master-receiver and the EEPROM is a slave-transmitter. In both cases, the master generates the clock, initiates the transaction, and terminates it.

Figure 191. I²C Bus Configuration Example

The I²C bus allows for a multi-master system, which means more than one device can initiate data transfers at the same time. To support this feature, the I²C bus arbitration relies on the wired-AND connection of all I²C interfaces to the I²C bus. Two masters can drive the bus simultaneously provided they are driving identical data. The first master to drive **SDA** high while another master drives **SDA** low loses the arbitration. The **SCL** line consists of a synchronized combination of clocks generated by the masters using the wired-AND connection to the **SCL** line.

The I²C bus serial operation uses an open-drain wired-AND bus structure, which allows multiple devices to drive the bus lines and to communicate status about events such as arbitration, wait states, error conditions and so on. For example, when a master drives the clock (**SCL**) line during a data transfer, it transfers a bit on every instance that the clock is high. When the slave is unable to accept or drive data at the rate that the master is requesting, the slave can hold the clock line low between the high states to insert a wait interval. The master's clock can only be altered by a slow slave peripheral keeping the clock line low or by another master during arbitration.

I²C transactions are either initiated by the IXP45X/IXP46X network processors as a master or are received by the IXP45X/IXP46X network processors as a slave. Both conditions may result in the processor doing reads, writes, or both to the I²C bus.

21.4.1 Operational Blocks

The I²C Bus Interface Unit is a slave peripheral device that is connected to the peripheral bus. The interrupt mechanism for the IXP45X/IXP46X network processors can be used for notifying the CPU that there is activity on the I²C bus. Polling can be



also be used instead of interrupts, although it would be very cumbersome. [Figure 190 on page 876](#) shows a block diagram of the I²C Bus Interface Unit and its interface to the internal bus.

The I²C Bus Interface Unit consists of the two wire interface to the I²C bus, an 8-bit buffer for passing data to and from the processor, a set of control and status registers, and a shift register for parallel/serial conversions.

The I²C Bus Interface Unit can initiate an interrupt when a buffer is full, buffer empty, slave address detected, arbitration lost, or bus error condition occurs. All interrupt conditions must be cleared explicitly by software. See [“I2C Status Register - ISR” on page 899](#) for details.

The I²C Data Buffer Register (IDBR) is an 8-bit data buffer that receives a byte of data from the shift register interface of the I²C bus on one side and parallel data from the processor's internal bus on the other side. The serial shift register is not user accessible.

The control and status registers are located in the I²C memory-mapped address space. The registers and their function are defined in [“Register Definitions” on page 896](#).

The I²C Bus Interface Unit supports fast mode operation of 400 Kbps. Fast-mode logic levels, formats, and capacitive loading, and protocols are exactly the same as the 100 Kbps, standard mode. Because the data setup and hold times differ between the fast and standard mode, the I²C has been designed to meet the faster, standard mode requirements for these two specifications. Refer to the I²C Bus Specification for details.

21.4.2 I²C Bus Interface Modes

The I²C Bus Interface Unit can be in different modes of operation to accomplish a transfer. [Table 277](#) summarizes the different modes.

Table 277. Modes of Operation

Mode	Definition
Master - Transmit	I ² C Bus Interface Unit acts as a master. Used for a write operation. I ² C Bus Interface Unit sends the data. I ² C Bus Interface Unit is responsible for clocking. Slave device will be in slave-receive mode
Master - Receive	I ² C Bus Interface Unit acts as a master. Used for a read operation. I ² C Bus Interface Unit receives the data. I ² C Bus Interface Unit is responsible for clocking. Slave device will be in slave-transmit mode
Slave - Transmit	I ² C Bus Interface Unit acts as a slave. Used for a read (master) operation. I ² C Bus Interface Unit sends the data. Master device will be in master-receive mode.
Slave - Receive (default)	I ² C Bus Interface Unit acts as a slave. Used for a write (master) operation. I ² C Bus Interface Unit receives the data. Master device will be in master-transmit mode.

While the I²C Bus Interface Unit is in idle mode (neither receiving or transmitting serial data), the unit defaults to Slave-Receive mode. This allows the interface to monitor the bus and receive any slave addresses that might be intended for the IXP45X/IXP46X network processors.



When the I²C Bus Interface Unit receives an address that matches the 7-bit address found in the I²C Slave Address Register (ISAR) or the General Call Address (00H), the interface will either remain in Slave-Receive mode or transition to Slave-Transmit mode. This is determined by the Read/Write (R/W#) bit (the least significant bit of the byte containing the slave address). If the R/W# bit is low, the master initiating the transaction intends to do a write and the I²C Bus Interface Unit will remain in Slave-Receive mode. If the R/W# is high, the initiating master wants to read data and the slave transitions to Slave-Transmit mode. Slave operation is further defined in [“Slave Operations” on page 889](#).

When the IXP45X/IXP46X network processors want to initiate a read or write on the I²C bus, the I²C Bus Interface Unit will transition from the default Slave-Receive mode to Master-Transmit mode. If the processor wants to write data, the interface remains in Master-Transmit mode after the address transfer has completed. (see [“START Condition” on page 880](#)) for START information). If the processor wants to read data, the I²C Bus Interface Unit will transmit the start address, then transition to Master-Receive mode. Master operation is further defined in [“Master Operations” on page 886](#).

21.4.3 Start and Stop Bus States

The I²C bus defines a transaction START and a transaction STOP bus state that are used at the beginning and end of the transfer of one to an unlimited number of bytes on the bus.

The IXP45X/IXP46X network processors use the START and STOP bits in the I²C Control Register (ICR) to:

- Initiate an additional byte transfer
- Initiate a START condition on the I²C bus
- Enable Data Chaining (repeated START)
- Initiate a STOP condition on the I²C bus

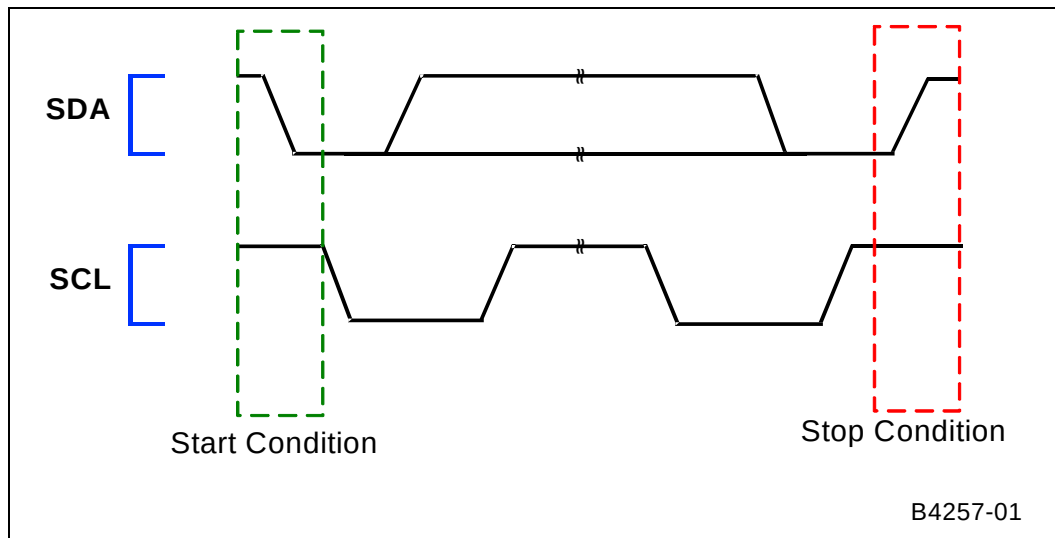
[Table 278](#) summarizes the definition of the START and STOP bits in the ICR.

Table 278. START and STOP Bit Definitions

STOP Bit	START Bit	Condition	Notes
0	0	No START or STOP	No START or STOP condition is sent by the I ² C Bus Interface Unit. This is used when multiple data bytes need to be transferred.
0	1	START Condition and Repeated START	The I ² C Bus Interface Unit will send a START condition and transmit the contents of the 8 bit IDBR after the START. The IDBR must contain the 7-bit address and the R/W# bit before a START is initiated. For a repeated start, the IDBR contents will contain the target slave address and the R/W# bit. This enables multiple transfers to different slaves without giving up the bus. The interface will stay in Master-Transmit mode if a write is used or transition to master-receive mode if a read is requested.
1	X	STOP Condition	In Master-Transmit mode, the I ² C Bus Interface Unit will transmit the 8-bit IDBR and then send a STOP on the I ² C bus. In Master-Receive mode, the Ack/Nack Control bit in the ICR must be changed to a negative Ack (see Section 21.5.3). The I ² C Bus Interface Unit will write the Nack bit (Ack/Nack Control bit must be 1), receive the data byte in the IDBR, then send a STOP on the I ² C bus.

[Figure 192](#) shows the relationship between the **SDA** and **SCL** lines for a START and STOP condition.

Figure 192. START and STOP Conditions



21.4.3.1 START Condition

The START condition (bits 1:0 of the ICR set to 2'b01) initiates a master transaction or repeated START. Software must load the target slave address and the R/W# bit in the IDBR (see “[I²C Data Buffer Register - IDBR](#)” on [page 902](#)) before setting the START ICR bit. The START and the IDBR contents are transmitted on the I²C bus when the ICR Transfer Byte bit is set. The I²C bus stays in master-transmit mode when a write is requested or enters master-receive mode when a read is requested. For a repeated start (a change in read or write or a change in the target slave address), the IDBR contains the updated target slave address and the R/W# bit. A repeated start enables multiple transfers to different slaves without giving up the bus.

The START condition is not cleared by the I²C unit. When arbitration is lost while initiating a START, the I²C unit may re-attempt the START when the bus becomes free. See “[Arbitration](#)” on [page 884](#) for details on how the I²C unit functions under those circumstances.

21.4.3.2 No START or STOP Condition

No START or STOP condition (bits 1:0 of the ICR set to 2'b00) is used in master-transmit mode while the processor is transmitting multiple data bytes (see [Figure 192](#)). Software writes the data byte, sets the IDBR Transmit Empty bit in the ISR (and interrupt when enabled), and clears the Transfer Byte bit in the ICR. The software then writes a new byte to the IDBR and sets the Transfer Byte ICR bit, which initiates the new byte transmission. This continues until the software sets the START or STOP bit. The START and STOP bits in the ICR are not automatically cleared by the I²C unit after the transmission of a START, STOP or repeated START.

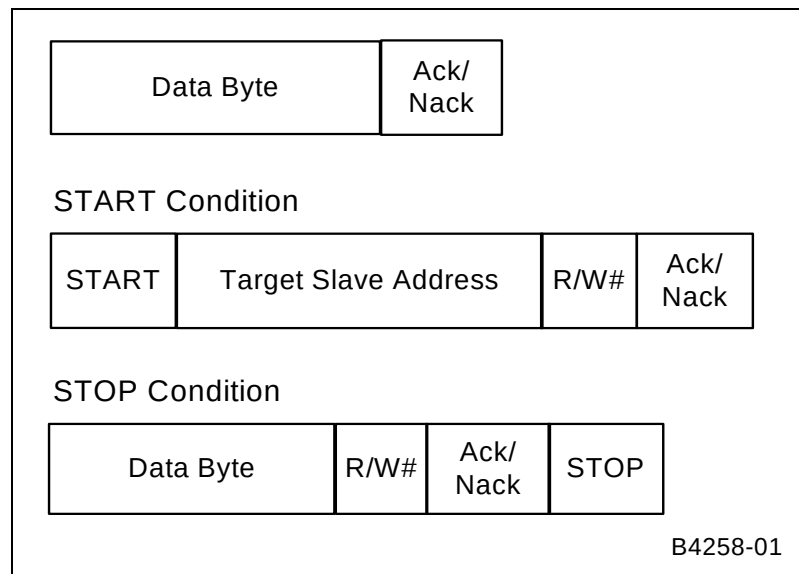
After each byte transfer (including the Ack/Nack bit) the I²C unit holds the **SCL** line low (inserting wait states) until the Transfer Byte bit in the ICR is set. This action notifies the I²C unit to release the **SCL** line and allow the next information transfer to proceed.



21.4.3.3 STOP Condition

The STOP condition (bits 1:0 of the ICR set to 2'b10) terminates a data transfer. In master-transmit mode, the STOP bit and the Transfer Byte bit in the ICR must be set to initiate the last byte transfer (see [Figure 192](#)). In master-receive mode, the processor must set the Ack/Nack bit, the STOP bit, and the Transfer Byte bit in the ICR to initiate the last transfer. Software must clear the STOP condition after it is transmitted.

Figure 193. START and STOP Condition Events



21.5 I²C Bus Operation

The I²C Bus Interface Unit transfers in 1 byte increments. A data transfer on the I²C bus always follows the sequence:

1. START
2. 7-bit Slave Address
3. R/W# Bit
4. Acknowledge Pulse
5. 8 Bits of Data
6. Ack/Nack Pulse
7. Repeat of Step 5 and 6 for Required Number of Bytes
8. Repeated START (Repeat Step 1) or STOP

21.5.1 Serial Clock Line (SCL) Generation

The I²C unit of the IXP45X/IXP46X network processors is required to generate the I²C clock output when in master mode (either receive or transmit). **SCL** clock generation is accomplished through the use of the ICCR value, which is programmed at initialization. [Equation 11](#) shows how the ICCR value is used to determine the **SCL** transition period:

Equation 11. SCL Transition Period

$$\text{SCL Transition Period} = (\text{ICCR Decimal Value} + 1) * (\text{processor-supplied clock period}) * (167 - (\text{Fast Mode Enable} * 125))$$



The **SCL** transition period is the amount of time the clock spends in the high or low state. When wait states are inserted or synchronization with another master is necessary, the I²C unit performs the necessary clock synchronization.

Note: The ICCR register is reserved on the IXP45X/IXP46X network processors. The I²C unit's master driver clock, SCL, has the option of running at 100-Kbps or 400-Kbps. This is set in the I²C Control Register (IDBR) (see [Section 21.10.1, "I²C Control Register - ICR" on page 897](#)).

21.5.2 Data and Addressing Management

Data and slave addressing is managed via the I²C Data Buffer Register (IDBR) and the I²C Slave Address Register (ISAR). The IDBR (see ["I²C Data Buffer Register - IDBR" on page 902](#)) contains data or a slave address and R/W# bit. The ISAR contains the processor's programmable slave address. Data coming into the I²C unit is received into the IDBR after a full byte is received and acknowledged. To transmit data, the CPU writes to the IDBR, and the I²C unit passes this onto the serial bus when the Transfer Byte bit in the ICR is set. See ["I²C Control Register - ICR" on page 897](#).

When the I²C unit is in master- or slave- transmit mode:

1. Software writes data to the IDBR over the internal bus.
This initiates a master transaction or sends the next data byte after the ISR[IDBR Transmit Empty] bit is set.
2. The I²C unit transmits the data from the IDBR when the ICR[Transfer Byte] is set.
3. When enabled, an IDBR Transmit Empty interrupt is signalled when a byte is transferred on the I²C bus and the acknowledge cycle is complete.
4. When the I²C bus is ready to transfer the next byte before the CPU has written the IDBR and a STOP condition is not in place, the I²C unit inserts wait states until the CPU writes a new value into the IDBR and sets the ICR[Transfer Byte] bit.

When the I²C unit is in master- or slave-receive mode:

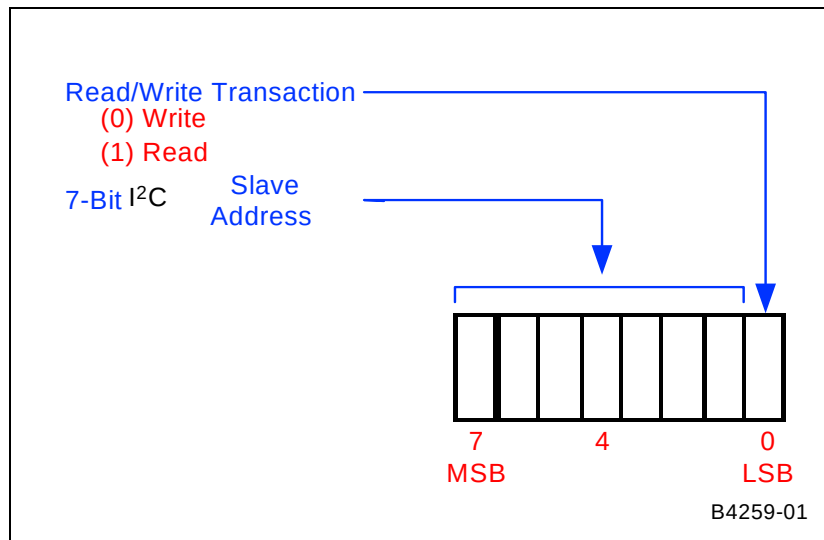
1. The processor reads IDBR data over the internal bus after the IDBR Receive Full interrupt is signalled.
2. The I²C unit transfers data from the shift register to the IDBR after the Ack cycle completes.
3. The I²C unit inserts wait states until the IDBR is read.
For acknowledge pulse information in receiver mode, see ["I²C Acknowledge" on page 883](#).
4. After the CPU reads the IDBR, the I²C unit writes the ICR[AckNak] Control bit and the ICR[Transfer Byte] bit, allowing the next byte transfer to proceed.

21.5.2.1 Addressing a Slave Device

As a master device, the I²C unit must compose and send the first byte of a transaction. This byte consists of the slave address for the intended device and a R/W# bit for transaction definition. The slave address and the R/W# bit are written to the IDBR (see ["I²C Data Buffer Register - IDBR" on page 902](#)).



Figure 194. Data Format of First Byte in Master Transaction



The first byte transmission must be followed by an Ack pulse from the addressed slave. When the transaction is a write, the I²C unit remains in master-transmit mode and the addressed slave device stays in slave-receive mode. When the transaction is a read, the I²C unit transitions to master-receive mode immediately following the Ack and the addressed slave device transitions to slave-transmit mode. When a Nack is returned, the I²C unit aborts the transaction by automatically sending a STOP and setting the ISR bus error bit.

When the I²C unit is enabled and idle (no bus activity), it stays in slave-receive mode and monitors the I²C bus for a START signal. Upon detecting a START pulse, the I²C unit reads the first seven bits and compares them to those in the I²C Slave Address Register (ISAR) and the general call address (00H). When the bits match those of the ISAR register, the I²C unit reads the eighth bit (R/W# bit) and transmits an Ack pulse. The I²C unit either remains in slave-receive mode (R/W# = 0) or transitions to slave-transmit mode (R/W# = 1). For actions when a general call address is detected, see “General Call Address” on page 891.

21.5.3 I²C Acknowledge

Every I²C byte transfer must be accompanied by an acknowledge pulse, which is always generated by the receiver (master or slave). The transmitter must release the **SDA** line for the receiver to transmit the acknowledge pulse (see Figure 195).

In master-transmit mode, when the target slave receiver device cannot generate the acknowledge pulse, the **SDA** line remains high. This lack of acknowledge (Nack) causes the I²C unit to set the bus error detected bit in the ISR and generate the associated interrupt (when enabled). The I²C unit aborts the transaction by generating a STOP automatically.

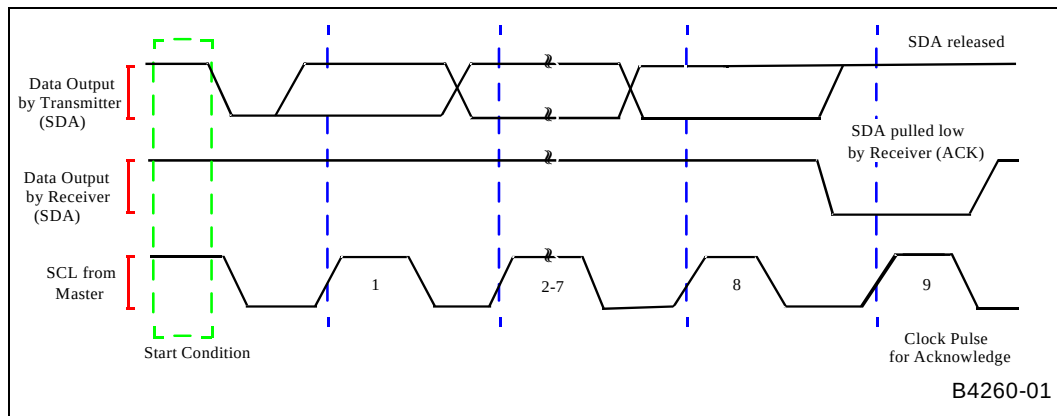
In master-receive mode, the I²C unit signals the slave-transmitter to stop sending data by using the negative acknowledge (Nack). The Ack/Nack bit value driven by the I²C bus is controlled by the Ack/Nack bit in the ICR. The bus error detected bit in the ISR is not set for a master-receive mode Nack (as required by the I²C bus protocol). The I²C unit automatically transmits the Ack pulse, based on the Ack/Nack ICR bit, after

receiving each byte from the serial bus. Before receiving the last byte, software must set the Ack/Nack Control bit to Nack. Nack is then sent after the next byte is received to indicate the last byte.

In slave mode, the I²C unit automatically acknowledges its own slave address, independent of the Ack/Nack bit setting in the ICR. As a slave-receiver, an Ack response is automatically given to a data byte, independent of the Ack/Nack bit setting in the ICR. The I²C unit sends the Ack value after receiving the eighth data bit of the byte.

In slave-transmit mode, receiving a Nack from the master indicates the last byte is transferred. The master then sends either a STOP or repeated START. The ISR's unit busy bit (2) will remain set until a STOP or repeated START is received.

Figure 195. Acknowledge on the I²C Bus



21.5.4 Arbitration

Arbitration on the I²C bus is required due to the multi-master capabilities of the I²C bus. Arbitration is used when two or more masters simultaneously generate a START condition within the minimum I²C hold time of the START condition.

Arbitration can continue for a long period. If the address bit and the R/W# are the same, the arbitration is then handled by the logic level of the data that is being driven. Due to the wired-AND nature of the I²C bus, no data is lost if both (or all) masters are outputting the same bus states. If the address, the R/W# bit, or the data are different, the master which outputted the high state (master's data will be different from **SDA**) will lose arbitration and shut its data drivers off. When losing arbitration, the I²C Bus Interface Unit will shut off the **SDA** or **SCL** drivers for the remainder of the byte transfer, set the Arbitration Loss Detected bit, then return to idle (Slave-Receive) mode.

21.5.4.1 SCL Arbitration

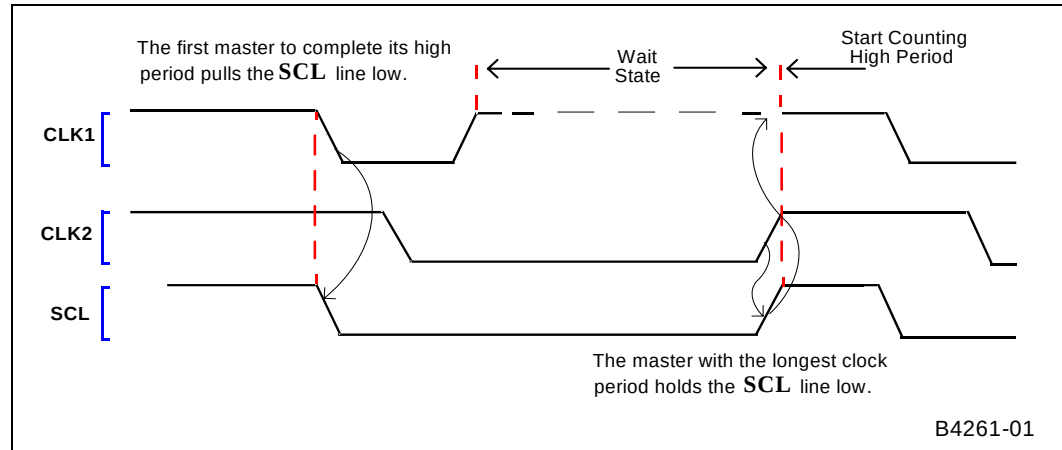
Each master on the I²C bus generates its own clock on the **SCL** line for data transfers. With masters generating their own clocks, clocks with different frequencies may be connected to the **SCL** line. Since data is valid when the clock is in the high period, a defined clock synchronization procedure is needed during bit-by-bit arbitration.

Clock synchronization is accomplished by using the wired-AND connection of the I²C interfaces to the **SCL** line. When a master's clock transitions from high to low, this causes the master to hold down the **SCL** line for its associated period (see Figure 196). The low to high transition of the clock may not change when another master has not completed its period. Therefore, the master with the longest low period holds down the



SCL line. Masters with shorter periods are held in a high wait-state during this time. Once the master with the longest period completes, the **SCL** line transitions to the high state, masters with the shorter periods can continue the data cycle.

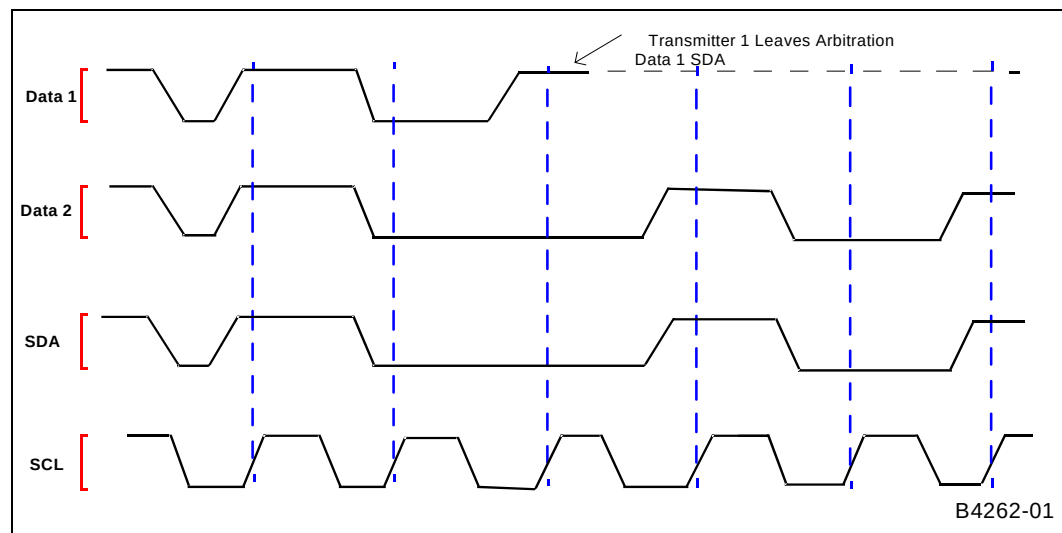
Figure 196. Clock Synchronization During the Arbitration Procedure



21.5.4.2 SDA Arbitration

Arbitration on the **SDA** line can continue for a long period starting with the address and R/W# bits and continuing with the data bits. [Figure 197](#) shows the arbitration procedure for two masters (more than two may be involved depending on how many masters are connected to the bus). When the address bit and the R/W# are the same, the arbitration is then handled by the logic level of the data that is being driven. Due to the wired-AND nature of the I²C bus, no data is lost when both (or all) masters are outputting the same bus states. When the address, R/W# bit, or data is different, the master that output the first high data bit loses arbitration and shuts its data drivers off. When the I²C unit loses arbitration, it shuts off the **SDA** or **SCL** drivers for the remainder of the byte transfer, sets the arbitration loss detected ISR bit, then returns to idle (Slave-Receive) mode.

Figure 197. Arbitration Procedure of Two Masters





When the I²C unit loses arbitration during transmission of the seven address bits and the processor is not being addressed as a slave device, the I²C unit re-sends the address when the I²C bus becomes free. A resend is possible because the IDBR and ICR registers are not overwritten when arbitration is lost.

If the I²C unit loses arbitration because another bus master addresses the processor as a slave device, the I²C unit switches to slave-receive mode and overwrites the original data in the I²C data buffer register. Software can clear the start and re-initiating the master transaction.

Note: Software must prevent I²C unit from starting a transaction to its own slave address because such a transaction puts I²C unit into an indeterminate state.

Boundary conditions exist for arbitration when an arbitration process is in progress and a repeated START or STOP condition is transmitted on the I²C bus. To prevent errors, the I²C unit, acting as a master, provides for the following sequences:

- No arbitration takes place between a repeated START condition and a data bit
- No arbitration takes place between a data bit and a STOP condition
- No arbitration takes place between a repeated START condition and a STOP condition

These situations arise only when different masters write the same data to the same target slave simultaneously and arbitration is not resolved after the first data byte transfer.

Note: Typically, software is responsible for ensuring arbitration is lost soon after the transaction begins. For example, the protocol might insist that all masters transmit their I²C address as the first data byte of any transaction ensuring arbitration is ended. A restart is then sent to begin a valid data transfer (the slave can then discard the master's address).

21.5.5 Master Operations

When software initiates a read or write on the I²C bus, the I²C unit transitions from the default slave-receive mode to master-transmit mode. The start pulse is sent followed by the 7-bit slave address and the R/W# bit. After the master receives an acknowledge, the I²C unit has the option of two master modes:

- Master-Transmit — The IXP45X/IXP46X network processors write data
- Master-Receive — The IXP45X/IXP46X network processors read data

The CPU initiates a master transaction by writing to the ICR register. Data is read and written from the I²C unit through the memory-mapped registers.

Table 279 describes the I²C Bus Interface Unit responsibilities as a master device.



Table 279. Master Transactions (Sheet 1 of 2)

I ² C Master Action	Mode of Operation	Definition
Generate clock output	Master-transmit Master-receive	- The master always drives the SCL line. - The ICCR register is written. - The SCL Enable bit must be set. - The Unit Enable bit must be set.
Write target slave address to IDBR	Master-transmit Master-receive	- The CPU writes to IDBR bits 7-1 before a START condition is enabled. - First 7 bits sent on bus after START. - See "Start and Stop Bus States" on page 879.
Write R/W# Bit to IDBR	Master-transmit Master-receive	- The CPU writes to the least significant IDBR bit with the target slave address. - If low, the master remains a master-transmitter. If high, the master transitions to a master-receiver. - See "Data and Addressing Management" on page 882.
Signal START Condition	Master-transmit Master-receive	- See "Generate clock output" above. - Performed after the target slave address and the R/W# bit are in the IDBR. - CPU sets the START bit. - CPU sets the Transfer Byte bit which initiates the start condition. - See "Start and Stop Bus States" on page 879.
Initiate first data byte transfer	Master-transmit Master-receive	- CPU writes byte to IDBR - I²C Bus Interface Unit transmits the byte when the Transfer Byte bit is set. - I²C Bus Interface Unit clears the Transfer Byte bit and sets the IDBR Transmit Empty bit when the transfer is complete.
Arbitrate for I ² C Bus	Master-transmit Master-receive	- If two or more masters signal a start within the same clock period, arbitration must occur. - The I²C Bus Interface Unit will arbitrate for as long as necessary. Arbitration takes place during slave address, R/W# bit, and data transmission and continues until all but one master loses the bus. No data is lost during arbitration. - If the I²C Bus Interface Unit loses arbitration, it will set the Arbitration Loss Detect ISR bit after byte transfer is complete and transition to slave-receive (default) mode. - If I²C Bus Interface Unit loses arbitration while attempting to send the target address byte, the I²C Bus Interface Unit will attempt to resend it when the bus becomes free. - The system designer must ensure the boundary conditions described in "I²C Bus Operation" on page 881 do not occur.
Write one data byte to the IDBR	Master-transmit only	- Data transmit mode of I²C master operation. - Occurs when the IDBR Transmit Empty ISR bit is set and the Transfer Byte bit is clear. If enabled, the IDBR Transmit Empty Interrupt is signalled to the IXP45X/IXP46X network processors. - CPU will write 1 data byte to the IDBR, set the appropriate START/STOP bit combination, and then set the Transfer Byte bit to send the data. Eight bits are written on the serial bus followed by a STOP if requested.
Wait for Acknowledge from slave-receiver	Master-transmit only	- As a master-transmitter, the I²C Bus Interface Unit will generate the clock for the acknowledge pulse. The I²C Bus Interface Unit is responsible for releasing the SDA line to allow slave-receiver Ack transmission. - See "I²C Acknowledge" on page 883.



Table 279. Master Transactions (Sheet 2 of 2)

I ² C Master Action	Mode of Operation	Definition
Read one byte of I ² C Data from the IDBR	Master-receive only	- Data receive mode of I²C master operation. - Eight bits are read from the serial bus, collected in the shift register then transferred to the IDBR after the Ack/Nack bit is read. - The CPU reads the IDBR when the IDBR Receive Full bit is set and the Transfer Byte bit is clear. If enabled, a IDBR Receive Full Interrupt is signalled to the CPU. - When the IDBR is read, if the Ack/Nack Status is clear (indicating Ack), the IXP45X/IXP46X network processors will write the Ack/Nack Control bit and set the Transfer Byte bit to initiate the next byte read. - If the Ack/Nack Status bit is set (indicating Nack), Transfer Byte bit is clear, STOP bit in the ICR is set, and Unit Busy bit in the ISR is set, then the last data byte has been read into the IDBR and the I²C Bus Interface Unit is sending the STOP. - If the Ack/Nack Status bit is set (indicating Nack), Transfer Byte bit is clear, but the STOP bit is clear, then the CPU has two options: 1. set the START bit, write a new target address to the IDBR, and set the Transfer Byte bit which will send a repeated start condition, 2. set the Master Abort bit and leave the Transfer Byte clear which will send a STOP only.
Transmit Acknowledge to slave-transmitter	Master-receive only	- As a master-receiver, the I²C Bus Interface Unit will generate the clock for the acknowledge pulse. The I²C Bus Interface Unit is also responsible for driving the SDA line during the Ack cycle. - If the next data byte is to be the last transaction, the CPU will set the Ack/Nack Control bit for Nack generation. - See "I2C Acknowledge" on page 883.
Generate a Repeated START to chain I ² C transactions	Master-transmit Master-receive	- If data chaining is desired, a repeated START condition is used instead of a STOP condition. - This occurs after the last data byte of a transaction has been written to the bus. - The CPU will write the next target slave address and the R/W# bit to the IDBR, set the START bit, and set the Transfer Byte bit. - See "Start and Stop Bus States" on page 879.
Generate a STOP	Master-transmit Master-receive	- Generated after the CPU writes the last data byte on the bus. - CPU generates a STOP condition by setting the STOP bit in the ICR. - See "Start and Stop Bus States" on page 879.

When the CPU needs to read data, the I²C unit transitions from slave-receive mode to master-transmit mode to transmit the start address and immediately following the ACK pulse transitions to master-receive mode to wait for the reception of the read data from the slave device (see [Figure 198](#)). It is also possible to have multiple transactions during an I²C operation such as transitioning from master-receive to master-transmit through a repeated start or Data Chaining (see [Figure 199](#)). [Figure 200](#) shows the wave forms of **SDA** and **SCL** for a complete data transfer.

Figure 198. Master-Receiver Read from Slave-Transmitter

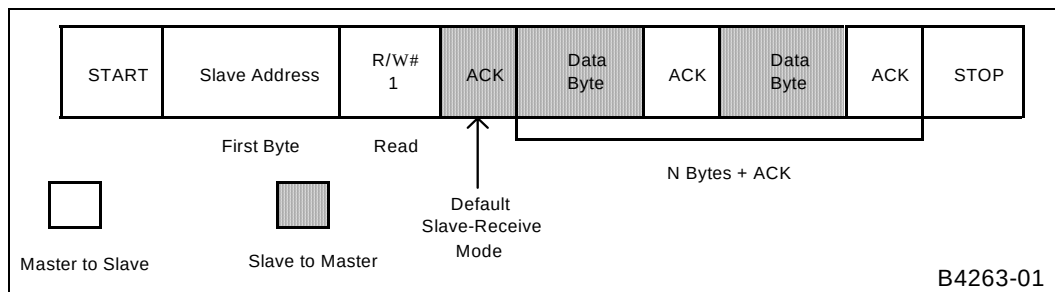




Figure 199. Master-Receiver Read from Slave-Transmitter / Repeated Start / Master-Transmitter Write to Slave-Receiver

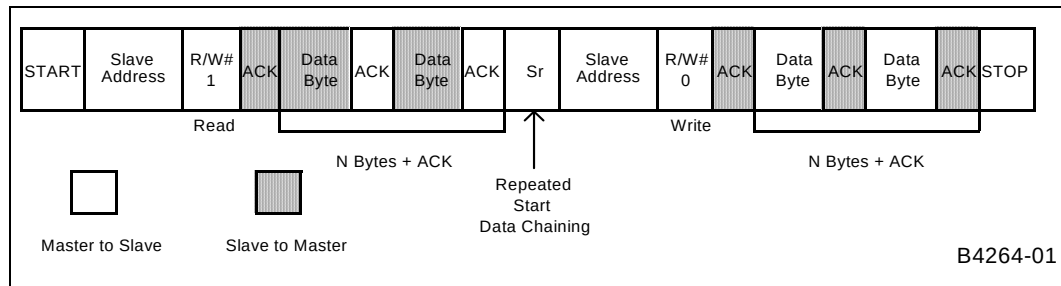
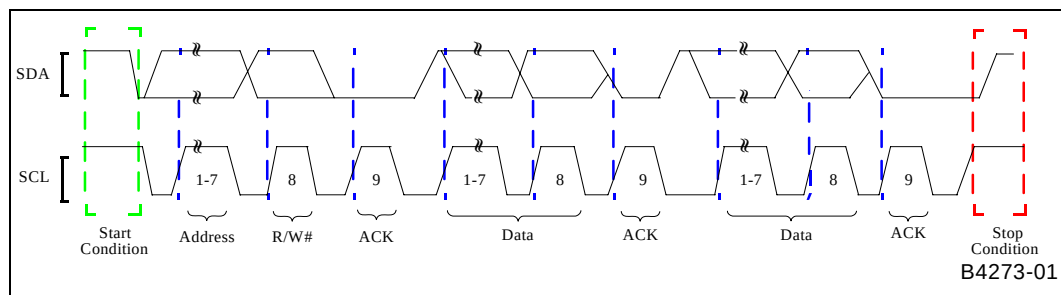


Figure 200. A Complete Data Transfer



21.5.6 Slave Operations

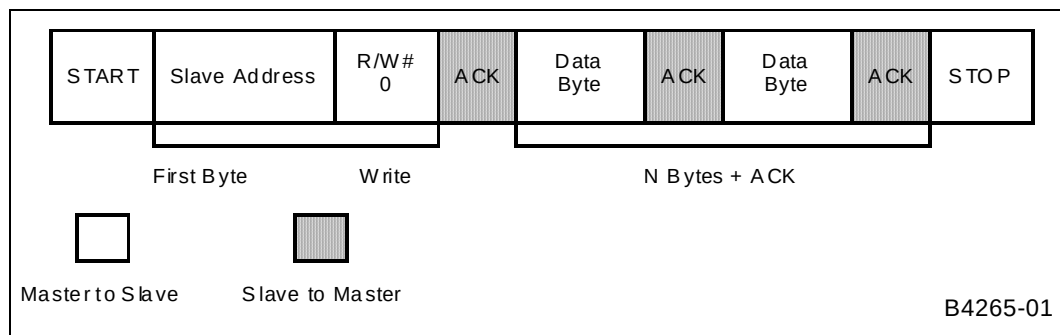
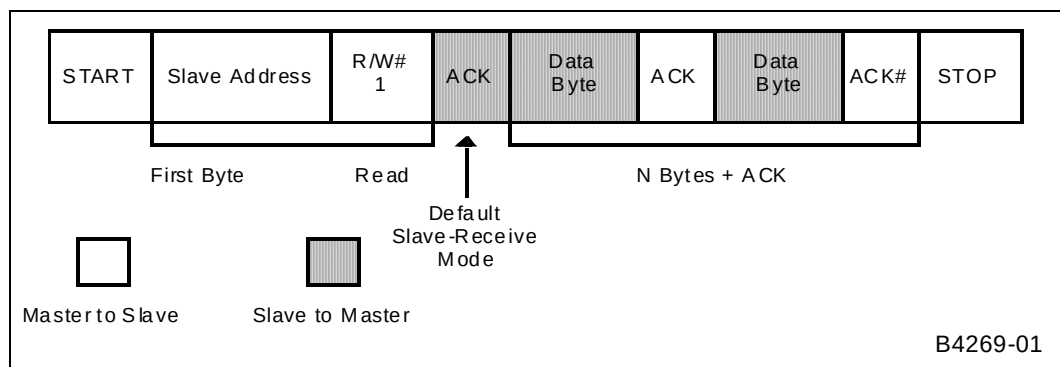
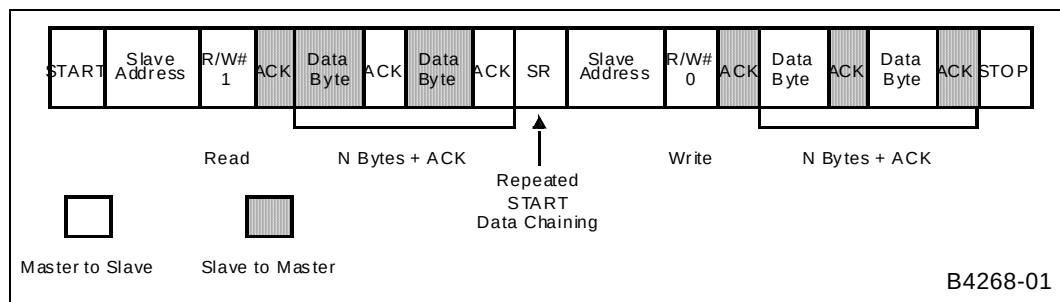
Table 280 describes the I²C Bus Interface Unit's responsibilities as a slave device.



Table 280. Slave Transactions

I ² C Slave Action	Mode of Operation	Definition
Slave-receive (default mode)	Slave-receive only	- I²C Bus Interface Unit monitors all slave address transactions. - The I²C Bus Interface Unit Enable bit must be set. - I²C Bus Interface Unit monitors bus for START conditions. When a START is detected, the interface reads the first 8 bits and compares the most significant 7 bits with the 7 bit I²C Slave Address Register and the General Call address (00H). If there is a match, the I²C Bus Interface Unit sends an Ack. - If the first 8 bits are all zero's, this is a general call address. If the General Call Disable bit is clear, both the General Call Address Detected bit and the Slave Address Detected bit in the ISR will be set. See "General Call Address" on page 891. - If the 8th bit of the first byte (R/W# bit) is low, the I²C Bus Interface Unit stays in slave-receive mode and the Slave Address Detected bit is cleared. If the R/W# bit is high, the I²C Bus Interface Unit transitions to slave-transmit mode and the Slave Address Detected bit is set.
Setting the Slave Address Detected bit	Slave-receive Slave-transmit	- Indicates the interface has detected an I²C operation that addresses the IXP45X/IXP46X network processors (this includes general call address). The IXP45X/IXP46X network processors can distinguish an ISAR match from a General Call by reading the General Call Address Detected bit. - An interrupt is signalled (if enabled) after the matching slave address is received and acknowledged.
Read one byte of I ² C Data from the IDBR	Slave-receive only	- Data receive mode of I²C slave operation. - Eight bits are read from the serial bus into the shift register. When a full byte has been received and the Ack/Nack bit has completed, the byte is transferred from the shift register to the IDBR. - Occurs when the IDBR Receive Full bit in the ISR is set and the Transfer Byte bit is clear. If enabled, the IDBR Receive Full Interrupt is signalled to the CPU. - CPU will read 1 data byte from the IDBR. When the IDBR is read, the IXP45X/IXP46X network processors will write the desired Ack/Nack Control bit and set the Transfer Byte bit. This causes the I²C Bus Interface Unit to stop inserting wait states and let the master transmitter write the next piece of information.
Transmit Acknowledge to master-transmitter	Slave-receive only	- As a slave-receiver, the I²C Bus Interface Unit is responsible for pulling the SDA line low to generate the Ack pulse during the high SCL period. - The Ack/Nack Control bit controls the Ack data the I²C Bus Interface Unit drives. See "I²C Acknowledge" on page 883.
Write one byte of I ² C data to the IDBR	Slave-transmit only	- Data transmit mode of I²C slave operation. - Occurs when the IDBR Transmit Empty bit is set and the Transfer Byte bit is clear. If enabled, the IDBR Transmit Empty Interrupt is signalled to the IXP45X/IXP46X network processors. - The IXP45X/IXP46X network processors will write a data byte to the IDBR and set the Transfer Byte bit to initiate the transfer.
Wait for Acknowledge from master-receiver	Slave-transmit only	- As a slave-transmitter, the I²C Bus Interface Unit is responsible for releasing the SDA line to allow the master-receiver to pull the line low for the Ack. - See "I²C Acknowledge" on page 883.

Figure 201 through Figure 203 are examples of I²C transactions. These show the relationships between master and slave devices.

**Figure 201. Master-Transmitter Write to Slave-Receiver****Figure 202. Master-Receiver Read to Slave-Transmitter****Figure 203. Master-Receiver Read to Slave-Transmitter, Repeated START, Master-Transmitter Write to Slave-Receiver**

21.5.7 General Call Address

A general call address is a transaction with a slave address of 00H. When a device requires the data from a general call address, it acknowledges the transaction and stays in slave-receiver mode. Otherwise, the device ignores the general call address. The other bytes in a general call transaction are acknowledged by every device that uses it on the bus. Devices that do not use these bytes must not send an Ack. The meaning of a general call address is defined in the second byte sent by the master-transmitter. Figure 204 shows a general call address transaction. The least significant bit (B) of the second byte defines the transaction. Table 281 shows the valid values and definitions when B=0.

If the I²C acts as a slave and receives a general call address while the ICR General Call Disable bit clears the I²C unit:



- Sets the ISR general call address detected bit
- Sets the ISR slave address detected bit
- Interrupts (when enabled) the IXP45X/IXP46X network processors

If the I²C unit receives a general call address and the ICR General Call Disable bit is set, the I²C unit ignores the general call address.

Figure 204. General Call Address

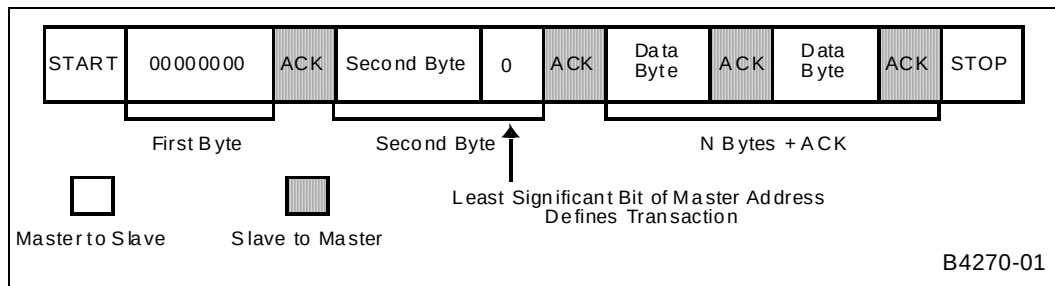


Table 281. General Call Address Second Byte Definitions

Least Significant Bit of Second Byte (B)	Second Byte Value	Definition
0	06H	2-byte transaction where the second byte tells the slave to reset and then store this value in the programmable part of their slave address.
0	04H	2-byte transaction where the second byte tells the slave to store this value in the programmable part of their slave address. No reset.
0	00H	Not allowed as a second byte
NOTE: Other values are not fixed and must be ignored.		

Software must ensure that the I²C unit is not busy before it asserts a reset. Software must also ensure that the I²C bus is idle when the unit is enabled after reset. When directed to reset, the I²C Bus Interface Unit will return to its default reset condition. The CPU is responsible for ensuring this occurs, not the I²C Bus Interface Unit hardware.

When B=1, the sequence is a hardware general call and is not supported by the I²C unit. Refer to the *I²C-Bus Specification* for information on hardware general calls.

I²C 10-bit addressing and CBUS compatibility are not supported.

21.6 Slave Mode Programming Examples

21.6.1 Initialize Unit

1. Write ISAR: Set slave address.
2. Write ICR: Enable all interrupts.
3. Set ICR[Unit Enable] bit to enable the I²C unit.

21.6.2 Write n Bytes as a Slave

1. When a Slave Address Detected interrupt occurs.



- Read ISR: Slave Address Detected (1), Unit Busy (1), R/nW# bit (1), Ack/Nack (0)
2. Write a 1 to the ISR[Slave Address Detected] bit to clear the interrupt.
3. Return from interrupt.
4. Load data byte to transfer in the IDBR.
5. Write ICR: Set Transfer Byte bit.
6. When a IDBR Transmit Empty interrupt occurs.
Read ISR: IDBR Transmit Empty (1), Ack/Nack (0), R/nW bit (0)
7. Load data byte to transfer in the IDBR.
8. Write ICR: Set Transfer Byte bit.
9. Write a 1 to the ISR[Transmit Empty] to clear interrupt.
10. Wait for interrupt.
Read ISR: Unit Busy (clear), Slave STOP Detected (set).
11. Repeat steps 6 to 10 for n-1 times. If, at any time, the slave does not have data, the I²C unit keeps SCL low until data is available.
12. When a IDBR Transmit Empty interrupt occurs.
Read ISR: IDBR Transmit Empty (1), Ack/Nak(1), R/nW bit (0).
13. Write a 1 to the ISR[Transmit Empty] bit to clear interrupt.
14. Return from interrupt.
15. When Slave Stop Detected interrupt occurs.
Read ISR: Unit Busy(0), Slave STOP Detected(1).
16. Write a 1 to the ISR[Slave Stop Detected] bit to clear interrupt.

21.6.3 Read n Bytes as a Slave

1. When a Slave Address Detected interrupt occurs.
Read ISR: Slave Address Detected(1), Unit busy (1), R/nW bit (0)
2. Write a 1 to the ISR[Slave Address Detected] bit to clear the interrupt.
3. Return from intrerrupt.
4. Write ICR: Set Transfer Byte bit to initiate the transfer
5. When an IDBR Receive Full interrupt occurs.
Read ISR: IDBR Receive Full (1), Ack/Nack (0), R/W# bit (0)
6. Read IDBR: To get the data.
7. Write a 1 to the ISR[Receive Full] bit to clear interrupt.
8. Return form interrupt.
9. Repeat steps 4 to 8 for n-1 times. Once the IDBR is full, the I²C unit will keep SCL low until the data is read.
10. Write ICR: Set Transfer Byte bit to release the I²C bus and allow next transfer.
11. When a Slave Stop Detected interrupt occurs.
Read ISR: Unit busy (0), Slave STOP Detected (1)
12. Write a 1 to the ISR[Slave STOP Detected] bit to clear the interrupt.



21.7 Master Programming Examples

21.7.1 Initialize Unit

1. Write ISAR: Set slave address.
2. Write ICR: Enable all interrupts (except Arb Loss), set **SCL** Enable, set Unit Enable and enable the I²C.

21.7.2 Write 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (0 for write).
2. Write ICR: Set START bit, Clear STOP bit, clear ICR[ALDIE], Set Transfer Byte bit to initiate the access.
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
Read status register: IDBR Transmit Empty (1), Unit Busy (1), R/nW bit (0).
4. Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.
5. Write a 1 to the ISR[ALD] bit if set.
If the master loses arbitration, it performs an address retry when the bus becomes free. The Arbitration Loss Detection interrupt is disabled to allow the address retry.
6. Load data byte to be transferred in the IDBR.
7. Write ICR: Clear START bit, Set STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit to initiate the access.
8. Wait for Buffer empty interrupt. When interrupt arrives (Note: Unit will be sending STOP):
Read status register: IDBR Transmit Empty (1), Unit busy (x), R/nW bit (0).
9. Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.
10. Clear ICR STOP bit.

21.7.3 Read 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (1 for read).
2. Write ICR: Set START bit, Clear STOP bit, Disable Arb loss interrupt, Set Transfer Byte bit to initiate the access.
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
Read ISR: IDBR Transmit Empty (1), Unit busy (1), R/nW bit (1).
4. Clear IDBR Transmit Empty bit to clear the interrupt.
5. Read byte with STOP
Write ICR: Clear START bit, Set STOP bit, Enable arb loss interrupt, Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access.
6. Wait for Buffer full interrupt. When interrupt arrives (Note: Unit will be sending STOP):
Read status register: IDBR Receive Full (set), Unit Busy (set - maybe), R/W# bit (Set), Ack/Nack bit (Set).
7. Clear IDBR Receive Full bit to clear the interrupt.
8. Read IDBR data.
9. Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional).



21.7.4 Write 2 Bytes and Repeated Start Read 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (0 for write).
2. Write ICR: Set START bit, Clear STOP bit, Set Transfer Byte bit to initiate the access.
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
Read status register: IDBR Transmit Empty (1), Unit busy (1), R/W# bit (0).
4. Clear IDBR Transmit Empty bit to clear the interrupt.
5. Load data byte to be transferred in the IDBR.
6. Initiate the write
Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit.
7. Wait for Buffer empty interrupt.
Read status register: IDBR Transmit Empty (1), Unit busy (1), R/W# bit (0).
8. Clear IDBR Transmit Empty bit to clear the interrupt.
9. Repeat steps 5-8 one time.
10. Load target slave address and R/nW bit in the IDBR. R/nW must be 1 for a read.
11. Send repeated start as a master.
Write ICR: Clear START bit, Clear STOP bit, clear ICR[ALDIE], Set Transfer Byte bit to initiate the access.
12. Wait for Buffer empty interrupt.
Read status register: IDBR Transmit Empty (1), Unit busy (1), R/nW bit (1).
13. Clear IDBR Transmit Empty bit to clear the interrupt.
14. Initiate the read.
Write ICR: Clear START bit, Set STOP bit, set ICR[ALDIE], Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access.
15. Wait for Buffer full interrupt. When interrupt comes (Note: Unit will be sending STOP).
Read status register: IDBR Receive Full (1), Unit busy (x), R/W# bit (1), Ack/Nack bit (1).
16. Clear IDBR Receive Full bit to clear the interrupt.
17. Read IDBR data.
18. Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional).

21.7.5 Read 2 Bytes as a Master — Send STOP Using the Abort

1. Write IDBR: Target slave address and R/W# bit (1 for read).
2. Write ICR: Set START bit, Clear STOP bit, Disable Arb loss interrupt, Set Transfer Byte bit to initiate the access.
3. Wait for IDBR Transmit Empty interrupt. When interrupt comes.
Read status register: IDBR Transmit Empty (1), Unit Busy (1), R/W# bit (1)
Clear IDBR Transmit Empty bit to clear the interrupt.
4. Read byte 1.
Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Clear Ack/Nack bit (Ack), Set Transfer Byte bit to initiate the access.
5. Wait for Buffer full interrupt.
Read status register: IDBR Receive Full (1), Unit busy (1), R/W# bit (1), Ack/Nack bit (0)



Clear IDBR Receive Full bit to clear the interrupt.
Read IDBR data.

6. Read byte 2 with Nack (STOP is not set because STOP or Repeated START will be decided on the byte read).
Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access.
7. Wait for Buffer full interrupt.
Read status register: IDBR Receive Full (1), Unit Busy (1), R/W# bit (1), Ack/Nack bit (1)
Clear IDBR Receive Full bit to clear the interrupt.
Read IDBR data.
There are now two options based on the byte read:
 - Send a repeated START.
 - Send a STOP only.Here, a STOP abort is sent.
Note: Had a NACK not been sent, the next transaction *must* involve another data byte read.
8. Send STOP abort condition. (STOP with no data transfer.)
Write ICR: Set Master abort.

21.8 Glitch Suppression Logic

The I²C Bus Interface Unit has built-in glitch suppression logic. Glitches will be suppressed according to: $2 * (ICCR + 1) * I^2C \text{ clock period}$. For example, with a 33 MHz (30-ns period) I²C clock, glitches of 60ns or less will be suppressed. This is within the 50-ns glitch suppression specification. Note, that for glitch suppression to be within specification, it is required that $(ICCR + 1) * I^2C \text{ clock period} > 25 \text{ ns}$. It is recommended that $(ICCR + 1) * I^2C \text{ clock period} = 30 \text{ ns}$.

21.9 Reset Conditions

Software is responsible for ensuring the I²C unit is not busy (ISR[3]) before asserting reset. Software is also responsible for ensuring the I²C bus is idle when the unit is enabled after reset. When directed to reset, the I²C unit returns to its default reset condition.

When the Unit Reset bit in the ICR is set, only the I²C unit of the IXP45X/IXP46X network processors resets and the associated I²C MMRs remain intact. When resetting the I²C unit with the ICR's unit reset, use the following guidelines:

1. In the ICR register, set the reset bit and clear the remainder of the register.
2. Clear the ISR register.
3. Clear reset in the ICR.

21.10 Register Definitions

The following registers are associated with the I²C Bus Interface Unit. They are all located within the peripheral memory-mapped address space of the IXP45X/IXP46X network processors.

**Table 282. Register Legend**

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 283. I²C Register Addresses

Register Addresses	Name	Description
0xC801_1000	ICR	"I2C Control Register - ICR" on page 897
0xC801_1004	ISR	"I2C Status Register - ISR" on page 899
0xC801_1008	ISAR	"I2C Slave Address Register - ISAR" on page 901
0xC801_100C	IDBR	"I2C Data Buffer Register - IDBR" on page 902
0xC801_1014	IBMR	"I2C Bus Monitor Register - IBMR" on page 902

21.10.1 I²C Control Register - ICR

The IXP45X/IXP46X network processors use the bits in the I²C Control Register (ICR) to control the I²C unit.

Register Name:						I ² C Control Register																													
Block Base Address:						0xC801_1000						Offset Address						Reg OffsetAddress						Reset Value						0x0000_0000					
Register Description:						I ² C Control Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																See table below.																			

Register		I ² C Control Register (Sheet 1 of 3)			
Bits	Name	Description	Reset Value	Access	
31:16	—	(Reserved)	0000H	—	
15	Fast Mode	Fast Mode: 0 = 100-Kbps operation 1 = 400-Kbps operation	0	RW	
14	Unit Reset	Unit Reset: 0 = No reset. 1 = Reset the I ² C unit only.	0	RW	
13	Slave Address Detected Interrupt Enable	Slave Address Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors upon detecting a slave address match or a general call address.	0	RW	



Register		I ² C Control Register (Sheet 2 of 3)		
Bits	Name	Description	Reset Value	Access
12	Arbitration Loss Detected Interrupt Enable	Arbitration Loss Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors upon losing arbitration while in master mode.	0	RW
11	Slave STOP Detected Interrupt Enable	Slave STOP Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors upon detecting a STOP condition while in slave mode.	0	RW
10	Bus Error Interrupt Enable	Bus Error Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors for the following I ² C bus errors: - As a master transmitter, no Ack was detected after a byte was sent. - As a slave receiver, the I²C unit generated a Nack pulse. Note: Software is responsible for guaranteeing that misplaced START and STOP conditions do not occur. See Section 21.8, "Glitch Suppression Logic" on page 896.	0	RW
09	IDBR Receive Full Interrupt Enable	IDBR Receive Full Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors when the IDBR has received a data byte from the I ² C bus.	0	RW
08	IDBR Transmit Empty Interrupt Enable	IDBR Transmit Empty Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the IXP45X/IXP46X network processors after transmitting a byte onto the I ² C bus.	0	RW
07	General Call Disable	General Call Disable: 0 = Enables the I ² C unit to respond to general call messages. 1 = Disables I ² C unit response to general call messages as a slave. This bit must be set when sending a master mode general call message from the I ² C unit.	0	RW
06	I ² C Unit Enable	I²C Unit Enable: 0 = Disables the unit and does not master any transactions or respond to any slave transactions. 1 = Enables the I ² C unit (defaults to slave-receive mode). Software must guarantee the I ² C bus is idle before setting this bit.	0	RW
05	SCL Enable	SCL Enable: 0 = Disables the I ² C unit from driving the SCL line. 1 = Enables the I ² C clock output for master mode operation.	0	RW
04	Master Abort	Master Abort: used by the I ² C unit when in master mode to generate a STOP without transmitting another data byte. 0 = The I ² C unit transmits STOP using the STOP ICR bit only. 1 = The I ² C unit sends STOP without data transmission. When in Master transmit mode, after transmitting a data byte, the ICR's Transfer Byte bit is cleared and IDBR Transmit Empty bit is set. When no more data bytes need to be sent, setting master abort bit sends the STOP. The Transfer Byte bit (03) must remain clear. In master-receive mode, when a Nack is sent without a STOP (STOP ICR bit was not set) and the IXP45X/IXP46X network processors do not send a repeated START, setting this bit sends the STOP. Once again, the Transfer Byte bit (03) must remain clear.	0	RW



Register		I ² C Control Register (Sheet 3 of 3)		
Bits	Name	Description	Reset Value	Access
03	Transfer Byte	Transfer Byte: used to send/receive a byte on the I ² C bus. 0 = Cleared by I ² C unit when the byte is sent/received. 1 = Send/receive a byte. The IXP45X/IXP46X network processors can monitor this bit to determine when the byte transfer has completed. In master or slave mode, after each byte transfer including Ack/Nack bit, the I ² C unit holds the SCL line low (inserting wait states) until the Transfer Byte bit is set.	0	RW
02	Ack/Nack Control	Ack / Nack Control: defines the type of Ack pulse sent by the I ² C unit when in master receive mode. 0 = The I ² C unit sends an Ack pulse after receiving a data byte. 1 = The I ² C unit sends a negative Ack (Nack) after receiving a data byte. The I ² C unit automatically sends an Ack pulse when responding to its slave address or when responding in slave-receive mode, independent of the Ack/Nack control bit setting.	0	RW
01	STOP	STOP: used to initiate a STOP condition after transferring the next data byte on the I ² C bus when in master mode. In master-receive mode, the Ack/Nack control bit must be set in conjunction with this bit. See Section 21.4.3.3, "STOP Condition" on page 881 for more details on the STOP state. 0 = Do not send a STOP. 1 = Send a STOP.	0	RW
00	START	START: used to initiate a START condition to the I ² C unit when in master mode. See Section 21.4.3.1, "START Condition" on page 880 for more details on the START state. 0 = Do not send a START. 1 = Send a START.	0	RW

21.10.2 I²C Status Register - ISR

I²C interrupts are signalled to the interrupt controller of the IXP45X/IXP46X network processors by the I²C Interrupt Status Register (ISR). Software uses the ISR bits to check the status of the I²C unit and bus. ISR bits (bits 9-5) are updated after the Ack/Nack bit has completed on the I²C bus.

The ISR is also used to clear interrupts signalled from the I²C Bus Interface Unit. These are:

- IDBR Receive Full
- IDBR Transmit Empty
- Slave Address Detected
- Bus Error Detected
- STOP Condition Detect
- Arbitration Lost

Register Name:						I ² C Status Register - ISR																													
Block Base Address:						0xC801_1004						Offset Address						Reg OffsetAddress						Reset Value						0x0000_0000					
Register Description:						I ² C Status Register - ISR																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																		See table below.																	



Register		ISR		
Bits	Name	Description	Reset Value	Access
31:1 1	—	(Reserved)	000000H	—
10	Bus Error Detected	Bus Error Detected: 0 = No error detected. 1 = The I ² C unit sets this bit when it detects one of the following error conditions: - As a master transmitter, no Ack was detected on the interface after a byte was sent. - As a slave receiver, the I²C unit generates a Nack pulse. Note: When an error occurs, I ² C bus transactions continue. Software must guarantee that misplaced START and STOP conditions do not occur. See Section 21.5.4, "Arbitration" on page 884 .	0	RW1C
09	Slave Address Detected	Slave Address Detected: 0 = No slave address detected. 1 = I ² C unit detected a 7-bit address that matches the general call address or ISAR. An interrupt is signalled when enabled in the ICR.	0	RW1C
08	General Call Address Detected	General Call Address Detected: 0 = No general call address received. 1 = I ² C unit received a general call address.	0	RW1C
07	IDBR Receive Full	IDBR Receive Full: 0 = The IDBR has not received a new data byte or the I ² C unit is idle. 1 = The IDBR register received a new data byte from the I ² C bus. An interrupt is signalled when enabled in the ICR.	0	RW1C
06	IDBR Transmit Empty	IDBR Transmit Empty: 0 = The data byte is still being transmitted. 1 = The I ² C unit has finished transmitting a data byte on the I ² C bus. An interrupt is signalled when enabled in the ICR.	0	RW1C
05	Arbitration Loss Detected	Arbitration Loss Detected: used during multi-master operation. 0 = Cleared when arbitration is won or never took place. 1 = Set when the I ² C unit loses arbitration.	0	RW1C
04	Slave STOP Detected	Slave STOP Detected: 0 = No STOP detected. 1 = Set when the I ² C unit detects a STOP while in slave-receive or slave-transmit mode.	0	RW1C
03	I ² C Bus Busy	I²C Bus Busy: 0 = I ² C bus is idle or the I ² C unit is using the bus (i.e., unit busy). 1 = Set when the I ² C bus is busy but the I ² C unit is not involved in the transaction.	0	RO
02	Unit Busy	Unit Busy: 0 = I ² C unit not busy. 1 = Set when the I ² C unit is busy. This is defined as the time between the first START and STOP.	0	RO
01	Ack/Nack Status	Ack / Nack Status: 0 = The I ² C unit received or sent an Ack on the bus. 1 = The I ² C unit received or sent a Nack. This bit is used in slave transmit mode to determine when the byte transferred is the last one. This bit is updated after each byte and Ack/Nack information is received.	0	RO
00	Read/Write Mode	Read / Write Mode: 0 = The I ² C unit is in master-transmit or slave-receive mode. 1 = The I ² C unit is in master-receive or slave-transmit mode. This is the R/W# bit of the slave address. It is automatically cleared by hardware after a stop state.	0	RO



21.10.3 I²C Slave Address Register - ISAR

The I²C Slave Address Register defines the I²C unit's 7-bit slave address to which the IXP45X/IXP46X network processors respond when in slave-receive mode. This register is written by the processor before enabling I²C operations. The register is fully programmable (no address is assigned to the I²C unit) so it can be set to a value other than those of hard-wired I²C slave peripherals that might exist in the system. The ISAR register default value is 0000000₂.

Register Name:						I ² C Slave Address Register - ISAR																													
Block Base Address:						0xC801_1008						Offset Address						Reg OffsetAddress						Reset Value						0x0000_007					
Register Description:						I ² C Slave Address Register - ISAR																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																								See table below.											

Register		I ² C Slave Address Register - ISAR															
Bits	Name	Description														Reset Value	Access
31:0 7	—	(Reserved)														000000H	—
6:0	I ² C Slave Address	I ² C Slave Address: The 7-bit address to which the I ² C unit responds when in slave-receive mode.														00H	RW



21.10.4 I²C Data Buffer Register - IDBR

The I²C Data Buffer Register is used by the IXP45X/IXP46X network processors to transmit and receive data from the I²C bus. The IDBR is accessed by the IXP45X/IXP46X network processors on one side and by the I²C shift register on the other. Data coming into the I²C Bus Interface Unit is received into the IDBR after a full byte has been received and acknowledged. Data going out of the I²C Bus Interface Unit is written to the IDBR by the processor core and sent to the serial bus.

When the I²C Bus Interface Unit is in transmit mode (master or slave), the processor writes data to the IDBR over the internal bus. This occurs when a master transaction is initiated or when the IDBR Transmit Empty Interrupt is signalled. Data is moved from the IDBR to the shift register when the Transfer Byte bit is set. The IDBR Transmit Empty Interrupt will be signalled (if enabled) when a byte has been transferred on the I²C bus and the acknowledge cycle is complete. If the IDBR is not written by the processor (and a STOP condition was not in place) before the I²C bus is ready to transfer the next byte packet, the I²C Bus Interface Unit will insert wait states until the processor writes the IDBR and sets the Transfer Byte bit.

When the I²C Bus Interface Unit is in receive mode (master or slave), the IXP45X/IXP46X network processors will read IDBR data over the internal bus. This occurs when the IDBR Receive Full Interrupt is signalled. The data is moved from the shift register to the IDBR when the Ack cycle is complete. The I²C Bus Interface Unit will insert wait states until the IDBR has been read. Refer to [Section 21.5.3, “I²C Acknowledge” on page 883](#) for acknowledge pulse information in receiver mode. After the processor reads the IDBR, the Ack/Nack Control bit is written and the Transfer Byte bit is written, allowing the next byte transfer to proceed on the I²C Bus. The IDBR register is 00H after reset.

Register Name:						I ² C Data Buffer Register - IDBR																													
Block Base Address:						0xC801_100C						Offset Address						Reg OffsetAddress						Reset Value						0x0000_0000					
Register Description:						I ² C Data Buffer Register - IDBR																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																							See table below.												

Register		I ² C Data Buffer Register - IDBR			
Bits	Name	Description	Reset Value	Access	
31:0 8	—	(Reserved)	000000H	—	
7:0	I ² C Data Buffer	Buffer for I ² C bus send/receive data.	00H	RW	

21.10.5 I²C Bus Monitor Register - IBMR

The I²C Bus Monitor Register (IBMR) tracks the status of the **SCL** and **SDA** pins. The values of these pins are recorded in this read-only register so that software may determine if the I²C bus is hung and the I²C unit must be reset.



Register Name:						I ² C Bus Monitor Register - IBMR																													
Block Base Address:						0xC801_1014						Offset Address						Reg OffsetAddress						Reset Value						0x0000_0000					
Register Description:						I ² C Bus Monitor Register - IBMR																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																																SCL		SDA	

Register		I ² C Bus Monitor Register - IBMR		
Bits	Name	Description	Reset Value	Access
31:2	—	(Reserved)	0	—
1	SCL Status	SCL Status: This bit continuously reflects the value of the SCL pin.	1	RO
0	SDA Status	SDA Status: This bit continuously reflects the value of the SDA pin.	1	RO





22.0 Public Key Exchange Crypto Engine

The Public Key Exchange (PKE) Crypto Engine is used to accelerate the establishment of secure connections by accelerating the key exchange process. This particular implementation consists of four main components:

- [Chapter 23.0, "AHB-PKE Bridge"](#)
- [Chapter 24.0, "Random Number Generator"](#)
- [Chapter 25.0, "Exponentiation Acceleration Unit"](#)
- [Chapter 26.0, "Hashing Unit \(SHA\)"](#)





23.0 AHB-PKE Bridge

23.1 Overview

The AHB-PKE Bridge consists of a protocol translation from the AHB bus to the PKE / Crypto Bus. The bridge is used to connect the high-speed AHB bus to the PKE Crypto Bus. The PKE Crypto Bus is where the hardware accelerator peripherals — such as the RNG, SHA, and EAU — reside.

The bridge provides a 32-bit-wide data path between the two buses. Address bits 31:17 of the AHB bus are not sent to the bridge. The AHB arbiter decodes the upper 8 bits of the address, and a Select signal is used to indicate that the address references a location downstream of the bridge, on the PKE bus.

Address bits 16:8 are decoded by the bridge and used to determine which of the crypto peripherals is selected. Address bits 11:1 are passed to the PKE Bus. All writes and reads to PKE peripherals must be word aligned single word accesses. The PKE bridge transforms AHB word accesses into two half-word accesses on the PKE local bus.

Only the Intel XScale® Processor bus master has access to the PKE peripherals. The AHB arbiter enforces this access rule.

23.2 Feature List

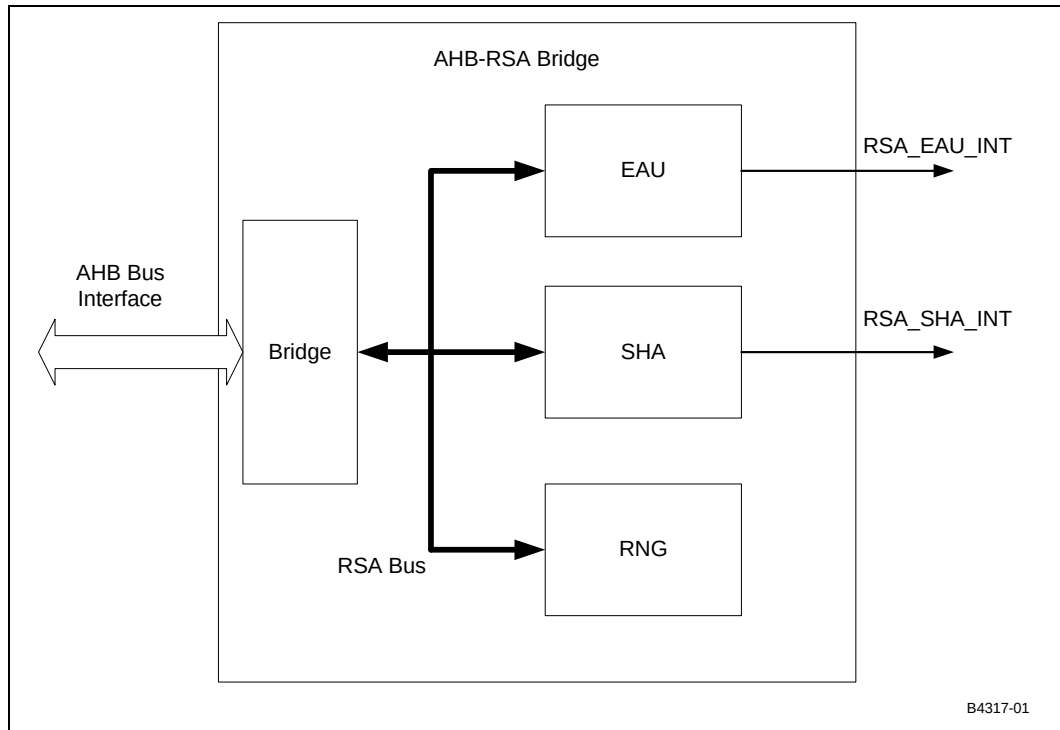
The AHB-PKE Bridge supports:

- 32-bit wide data path only - Single Word accesses only, no byte, half word or burst accesses supported, all other size accesses return AHB Bus ERROR
- Only supports SINGLE word reads and writes, all other accesses will return AHB Bus ERROR.
- 10 bits of address downstream
- Decode of address bits 16:8 to select a peripheral on the PKE bus
- Support up to a 4KB EAU ram space
- Additionally the EAU and SHA blocks will generate interrupts to the Intel XScale processor. (eau_int, sha_int)



23.3 Block Diagram

Figure 205. AHB-PKE Bridge Block Diagram



23.4 Theory of Operation

The function of the AHB-PKE Bridge is to pass data between the Intel XScale processor AHB master and PKE Bus slaves.

23.4.1 Peripheral Information

Each PKE peripheral is selected by a PKE_sel_xxx signal that is active whenever an address is within the peripheral's address space. This signal is held active for the duration of an PKE bus cycle. Each PKE peripheral drives data on the xxx_rdata signal while the xxx_rdy signal is active. Table 284 lists the PKE peripherals and their location within the address space of the PKE bus and the name of the corresponding sel and rdata signals.

Table 284. PKE Peripheral Memory Map and Access Information

Address Range	Peripheral	PSEL name	PRDATA signal name
7000_0000 – 7000_07FF	EAU RAM 2 Kbyte	PKE_sel_eau_ram	eau_rdata
7000_0800- 7000_1FFF	Reserved for EAU RAM expansion 6 Kbyte		
7000_2000 – 7000_00FF	EAU MMR	PKE_sel_eau_mmr	eau_rdata
7000_2100	RNG MMR	PKE_sel_rng	rng_rdata



Table 284. PKE Peripheral Memory Map and Access Information

Address Range	Peripheral	PSEL name	PRDATA signal name
7000_2101 – 7000_01FF	(Reserved)	N/A	N/A
7000_2200 – 7000_22FF	SHA MMR	PKE_sel_sha	sha_rdata
7000_2300 – 7000_23FF	(Reserved)	N/A	N/A
7000_2400 – 7000_24FF	(Reserved)	N/A	N/A
7000_2500 – 7FFF_FFFF	(Reserved)	N/A	N/A





24.0 Random Number Generator

24.1 Theory of Operation

The Random Number Generator (RNG) unit provides a digital, random-number-generation capability.

The Random Number Generator (RNG) produces a series of random numbers for use in, among other things, key generation. The only method of access to these numbers is via a read from the random number FIFO. No other visibility is provided into the workings of this unit by the processor, nor is there any method of control or parameterization by any external method.

24.2 Detailed Register Descriptions

Table 285. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 286. Register Summary

Block Address	Offset Address	Register Name	Description	Reset Value	Page Number
0x7000	_2100	RNG_FIFO	FIFO containing random numbers generated by the RNG. After a number is read, it is removed from the stack, and the next number is made available	Unknown	912



24.2.1 Registers

24.2.1.1 Random Number FIFO

Register Name:						RNG_FIFO																											
Block Base Address:						0x7000						Offset Address						_2100						Reset Value						Unknown			
Register Description:						FIFO containing random numbers generated by the RNG. After a number is read, it is removed from the stack, and the next number is made available																		Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
RNG_FIFO																																	

Register		RNG_FIFO															
Bits	Name	Description												Reset Value		Access	
31:0	Random Number	Number at top of random number FIFO stack.												Unknown		RO	

24.3 Power-Management Requirements

The RNG implements no power management features.

24.4 Reset

On Power-On reset, all internal flip-flops — with the exception of the random-number FIFO — are initialized. The pointers to the random number FIFO are, however, initialized, effectively “emptying” the random number FIFO.

After reset clears, the unit initializes itself prior to beginning to fill the random number FIFO. No read request will be fulfilled until at least one word has been loaded into the random number FIFO.

24.5 Error / Abnormal Conditions

In the event of a FIFO underflow, the RNG will not assert the `rng_rdy` signal, thus preventing the completion of a read operation, until a new word is available. This may result in longer access times under certain operational conditions.



25.0 Exponentiation Acceleration Unit

25.1 Overview

The Exponentiation Acceleration Unit (EAU) supports various large-number arithmetic operations. These operations include modular exponentiation, modular reduction, multiply, add and subtract. These operations are controlled through a set of memory mapped registers. Parameters for and results of the operations are written in little-endian ordering into a RAM (contained within the EAU) which the EAU state machine accesses and also uses for temporary registers. The arithmetic operations supported by the EAU are used by software executing in the Intel XScale® Processor to build larger cryptographic functions such as signing and verification procedures. Since the EAU executes only one operation at a time, the Intel XScale processor must serialize the required operations to the EAU.

The EAU begins operating after the Intel XScale processor has moved data into the EAU RAM and loads the EAU's command register with an appropriate command. After executing the command, the EAU appropriately sets its status bits and waits idle until it receives another command from the Intel XScale processor.

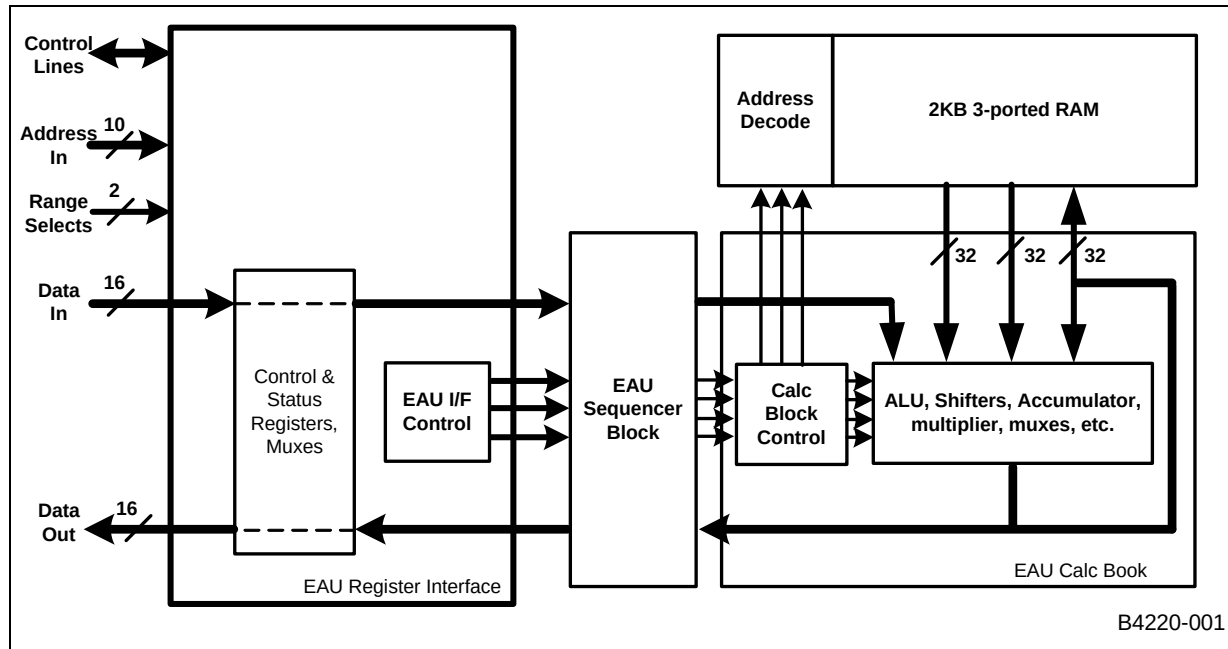
25.2 Feature List

- Speeds-up large number arithmetic operations
- Modular Exponentiation: Function is $C = M^e \bmod N$ where M , e , N are up to 2,048 bits
- Modular Reduction: Function is $R = A \bmod N$ where A is up to 4,096 bits and N is up to 2,048 bits
- Large Number Multiply: Function is $R = A * B$ where A & B are up to 2,048 bits
- Large Number Add: Function is $R = A + B$ where A & B are up to 2,048 bits
- Large Number Subtract: Function is $R = A - B$ where A & B are up to 2,048 bits
- Supports modulus operands of 256 / 512 / 1024 / 1536 / 2048 bits long. Support for 256-bit operands is provided so that the Chinese Remainder Theorem (CRT) algorithm can be used for 512-bit values.
- Modular Exponentiation time can be made *independent* of exponent value (Fast Exponent configuration bit). This may help against timing attacks on the unit.
- Modular Exponentiation time can be reduced by ignoring leading zeroes (Short Exponent configuration bit).
- Word-readable, word-writable registers
- Parameter RAM is readable and *word*-writable, through a internal register and state machine.



25.3 Block Diagram

Figure 206. Exponentiation Acceleration Unit: Block Diagram



25.3.1 Operand Restrictions

Since the EAU implements modular arithmetic functions using a fixed-size datapath ALU, there are some assumptions / restrictions for the operands that must be observed. See Table 287 for details.

Table 287. EAU Operand Size Restrictions and Assumptions

Operation	Operand Assumptions	Result (R) Size
Addition $R = A + B$	A, B must be same size (k bits) A, B can be padded with leading zeros out to k bits	k+1 bits
Subtraction $R = A - B$	A, B must be same size (k bits) A, B can be padded with leading zeros out to k bits	k bits
Multiplication $R = A * B$	A, B must be same size (k bits) A, B can be padded with leading zeros out to k bits	2k bits
Modular Reduction $R = A \bmod N$	size of A = 2k bits, size of N = k bits MSB of N = 1, LSB of N can be 0 or 1 Note: A can be padded with leading zeros.	k bits
Modular Exponentiation $R = M^e \bmod N$	M, e, N must be same size (k bits) Modulus N is a k-bit number, i.e., $2^{k-1} < n < 2^k$ MSB & LSB of N = 1 Value of M < N Note: e can be padded with leading zeros.	k bits
Modular Inverse $R = A^{-1} \bmod N$	A, N must be same size (k bits) Value of A < N	k bits
Note: k = size of modulus in bits		



25.3.2 EAU RAM Writes

Writes to the EAU RAM are buffered. The buffering occurs on a word (32-bit) basis. EAU RAM writes signal completion of a write using the same signaling protocol described for EAU RAM Reads.

25.3.3 EAU RAM Reads

EAU RAM Reads are done indirectly: in response to a read request, a state machine fetches the data from the RAM and places it in a buffer. This takes several clock cycles. The EAU coordinate the return of the read data to the requester via the RSA bridge.

25.4 Detailed Register Descriptions

This section describes the MMR registers that are accessible in the EAU.

Table 288. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 289. Register Summary

Block Address	Offset Address	Register Name	Description	Reset Value	Page
0x7000	_2000	EAUCMD	EAU Command Reg	0x0001_0000	915
0x7000	_2004	EAUSTAT	The EAU Status Register is a single byte used to read the status of the EAU	0x0000_0000	917
0x7000	_2008	EAUCNT	EAU Count Register	0x0000_0000	917
0x7000	_200C	EAUINT	Interrupt Register	0x0000_0000	918

25.4.1 EAU Command Register

Register Name:					EAUCMD																										
Block Base Address:			0x7000					Offset Address					_2000					Reset Value					0x0001_0000								
Register Description:					The EAU Command Register is a 32-bit register (read and written as four byte operations) used to control the EAU functionality. It can be written only when the EAU is not active.															Access:			(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Rsv'd)		SE	FE	CMD				(Rsv'd)	SIZE					(Rsv'd)	DST			(Rsv'd)	SRC			FILL									



Register		EAU Command Register		
Bits	Name	Description	Reset Value	Access
31:30		(Reserved)	0x00	
29	SE	EAU Short Exponent Enabled 0: do NOT skip leading zeros in modular exponentiation operation 1: skip leading zeros in modular exponentiation operation (overrides FE=0)	0x0	RW
28	FE	EAU Fast Exponent Enabled 0: do not skip zeros in modular exponentiation operation 1: skip ALL zeros in modular exponentiation operation	0x0	RW
27:24	CMD	EAU Command Bits have special encoding for EAU commands. EAU starts execution when the high byte of the Control Register is written. '0000 NOP/Idle '0001 Modular Exponential ($C = Me \bmod n$) '0010 Large Number Multiplication ($A * B$) '0011 Large Number Addition ($A + B$) '0100 Modular Reduction ($A \bmod n$) '0101 Large Number Subtraction ($A - B$) '0110 Clear EAU RAM '0111 Reserved '1000 MonPro ($R = a\_bar * b\_bar * r - 1 \bmod n$) '1001 Reserved for EAU RAM reads '1010 Reserved for EAU RAM writes '1011 RAM Fill (Reg Dst < FILL) '1100 EAU RAM Register Clear (Reg Dst < 0) '1101 EAU RAM Register Copy (Reg Dst < Reg Src) '1110 Reserved for RAM BIST '1111 Reserved	0x000	RW
23		(Reserved)	0x0	
22:16	SIZE	Operand Size bits: This field defines the length of operands to be operated on (# of 32-bit words). Valid values for these bits are 8, 16, 24, 32, 40, 48, 56 and 64. The value of 2 is illegal due to architectural pipeline limitations. This field is applicable to Modular Exponentiation, Multiplication, Modular Reduction, Add, Subtract, Copy, Clear, and Fill commands.	0x1	RW
15		(Reserved)	0x0	
14:12	DST	EAU Destination Field This field contains the destination register number for copy, clear, and fill commands.	0x0	RW
11		(Reserved)	0x0	
10:8	SRC	EAU Source Field This field contains the source register number for copy commands.	0x0	RW
7:0	FILL	EAU Fill Value This byte contains the value to be written into the EAU RAM using the FILL command. It fills the EAU register defined in the DST field of the EAU Command Register. The number of double words that are filled is defined in the SIZE field of the EAU Command Register.	0x0	RW



25.4.2 EAU Status Register

Register Name:						EAUSTAT																										
Block Base Address:			0x7000					Offset Address					_2004					Reset Value					0x0000_0000									
Register Description:						The EAU Status Register is a single byte used to read the status of the EAU.																			Access:			(See below.)				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
(Reserved)																									EQ	CB	RSVD	DONE	ES			

Register		EAU Status Register		
Bits	Name	Description	Reset Value	Access
31:5		(Reserved)	0x00	
4	EQ	EAU Equal Bit 0: indicates that the operands of a subtract operation are NOT equal 1: indicates that the operands of a subtract operation are equal	0x00	RO
3	CB	EAU Carry/Borrow Bit: this is a conventional 2's-complement-arithmetic carry/borrow bit that is set or cleared by an add or subtract operation. The state of this bit after any other operation is implementation dependent. 0: indicates there was no Carry/Borrow as a result of a add/sub operation 1: indicates there was a Carry/Borrow as a result of a add/sub operation	0x00	RO
2		(Reserved)	0x00	
1	DONE	Indicates completion of operation when set, write a zero to clear.	0x00	RW
0	ES	EAU Status Bit 0: EAU is currently inactive 1: EAU is currently active	0x00	RO

25.4.3 EAU Count Register

Register Name:					EAUCNT																										
Block Base Address:					0x7000					Offset Address					_2008					Reset Value					0x0000_0000						
Register Description:					The 32-bit EAU Count register can be read to determine the number of clocks used by the previous EAU operation. It starts counting (from its current value) at the start of the operation and stops when the operation is complete. This register should be read immediately after the operation is completed. Once read, the register is automatically cleared (triggered by a read of the most significant byte). This register exists to support testing.															Access:			(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CountValue																															



Register		EAU Count Register		
Bits	Name	Description	Reset Value	Access
31:0	COUNT	Contains the previous value of [COUNT (prior to the start of the last EAU operation) + the number of EAU cycles for the last EAU operation]. This register is cleared automatically when a new command is started.	0x00	RO

25.4.4 EAU Interrupt Register

Register Name:						EAUINT																													
Block Base Address:						0x7000						Offset Address						_200C						Reset Value						0x0000_0000					
Register Description:						The EAU Interrupt Register is a single bit used to read and clear the EAU interrupt.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																																INT			

Register		EAU Status Register			
Bits	Name	Description	Reset Value	Access	
31:16		(Reserved)	0x00		
15:1		(Reserved)	0x00		
0	INT	EAU InterruptBit 0: EAU Interrupt is currently inactive 1: EAU Interrupt is currently active. Write 1 to clear	0x0	RW	

25.4.5 EAU RAM Registers

The 2-Kbyte EAU RAM can be read and written with certain restrictions. While the EAU is active, all requests for reads and writes to the RAM will be ignored. The RAM will not be reset after a power-on or pin reset assertion. The content of the RAM is indeterminate until they are initialized by the CPU / Intel XScale processor.

Writes to the RAM are all 32-bit, and are triggered by a half-word write to a RAM address that is the high half of the word. The required usage model is to write a 4-byte word two bytes at a time from least significant half-word to most significant half-word. In actual implementation, the register interface stores the bottom two bytes in a write buffer until the two high bytes are written. Repeated high half-word writes (at other addresses) will initialize other 32-bit words with half of the writes required to initialize all bytes individually. Do not attempt to write a byte to the RAM: byte writes trigger a pre-read operation, as described below, and the data in the write buffer is ignored.

Reads to the RAM are also buffered and accessed with different timing than reads to registers. Reading values from the EAU takes several cycles, and during this time the Intel XScale processor is held off. using EAU_DONE. The EAU RAM resides at memory locations as shown in [Table 290, "EAU RAM Memory Locations" on page 919](#).



Table 290. EAU RAM Memory Locations

Address Offset from EAU RAM Base (0x7000_0000)	Modular Exponentiation ¹ (C=Me mod n)	Modular Reduction ² (R = A MOD n)	Big number Multiplication (R=A*B)	Big Number Add, Sub (R=A op B)	MonPro ³ (R= a_bar * b_bar * r-1 mod n)	Copy, Clear, Fill
700H - 0FFFH	unused	unused	unused	unused	unused	Reg7 ⁴
600H - 6FFH	unused	unused	unused	unused	unused	Reg6 ⁴
500H - 5FFH	Modulus (n)	Modulus (n)	unused	unused	Modulus (n)	Reg5
400H - 4FFH	temp	temp	unused	unused	temp	Reg4
300H - 3FFH	Op1/Result (M/C)	A_hi	Result_hi	unused	Modulus (n)	Reg3
200H - 2FFH	Temp	A_lo/Result	Result_lo	Result	b_bar/Result	Reg2
100H - 1FFH	Temp	temp	B	B	a_bar	Reg1
000H - 0FFH	Exponent (e)	temp	A	A	unused	Reg0
Notes: 1. The Modular Exponentiation operation requires that the MSB and LSB of the modulus be 1. 2. The Modular reduction operation requires that the MSB of the modulus be 1. 3. MonPro is used for testing only. 4. The top two registers are used to temporarily store parameters for algorithms based on the Chinese Remainder Theorem (CRT).						

25.5 Performance Requirements

The equation for calculating the # of clock cycles for an exponentiation is given below:

Total clocks = (size of exponent) x (number of modular multiplies) x (number of long multiplies per modular multiply) x (number of clocks for each long multiply). Using a 32-by-32-bit, single-cycle multiplier, it takes 1,024 (32 * 32) clocks to multiply two 1,024-bit numbers.

For an operand length of 1024 bits the number of clocks needed for an exponentiation operation is: (exp) * (# Mod Mults) * (# Long Mults) * (# clks per long mult) or specifically;

$$1,024 * 2 * 3 * 1024 = 6,291,456 \text{ clocks @ } 133 \text{ MHz} = 47 \text{ ms for a worst case,}$$

If statistically half the exponent bits are 1, then an exponentiation operation would be approximately:

$$1,024 * 1.5 * 3 * 1,024 = 4,718,592 @ 133 \text{ MHz} = 35 \text{ ms}$$

Public key operations (short exponent) are approximated by:

$$3 * 2 * 3 * 1,024 = 18,432 \text{ clocks @ } 133 \text{ MHz} = 0.14 \text{ ms}$$

Note: This does not include Montgomery operations on the operands, which require additional clocks.





26.0 Hashing Unit (SHA)

26.1 Overview

The hashing unit (SHA) is used to accelerate digital signature computations and to whiten the RNG output. It is the responsibility of the Intel XScale® Processor to get the data from the RNG, provide it to the SHA unit, and retrieve the data after it is complete.

26.2 Feature List

The SHA wrapper supports:

- 32-bit wide data path
- 10 bits of address downstream
- Decode of address bits 3:0 for instruction to the hashing coprocessor
- A busy signal to the AHB/PKE bridge to signal that a hashing process is under way and cannot take any write or read commands
- An interrupt signal to signal the hashing process is completed

26.3 Block Diagram

26.4 Detailed Register Descriptions

Table 291. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 292. Hashing Coprocessor: Register Summary (Sheet 1 of 2)

Block Address	Offset Address	Register Name	Description	Reset Value	Page Number	Access
0x7000_	2200	Hash_Config	Hash Configuration Register	0x00000000	922	RW
0x7000_	2204	Hash_Do	Hash Do Register	N/A	922	WO



Table 292. Hashing Coprocessor: Register Summary (Sheet 2 of 2)

Block Address	Offset Address	Register Name	Description	Reset Value	Page Number	Access
0x7000_	2208	Hash_Int	Hash Interrupt Register	0x00000000	923	RW
0x7000_	220C	Hash_Chain	Hash Chain Register	0x00000000	923	RW
0x7000_	2210	Hash_Data	Hash Data FIFO	0x00000000	924	RW

26.4.1 Hash Configuration Register

Register Name:						Hash_Config																														
Block Base Address:						0x7000_						Offset Address						2200						Reset Value						0x00000000						
Register Description:						Hashing Coprocessor Configuration Register																				Access:				(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
WEW			REV			WED			RED			(Reserved)						MODE	(Reserved)																	

Register		Hash_Config		
Bits	Name	Description	Reset Value	Access
31:3 0	WEW	Set endianness for writing the chaining variables.	0	RW
29:2 8	REV	Set endianness for reading the chaining variables.	0	RW
27:2 6	WED	Set endianness for writing the data store.	0	RW
25:2 4	RED	Set endianness for reading the data variables.	0	RW
23:1 7	(Reserved)	These bits are always 0.	0	RW
16	MODE	SHA-1 or MD5 mode select 1 - MD-5 0 - SHA-1	0	RW
15:0	(Reserved)	These bits are always 0.	0	RW

26.4.2 Hash Do Register

Register Name:						Hash_Do																													
Block Base Address:						0x7000_						Offset Address						2204						Reset Value						N / A					
Register Description:						Hashing Coprocessor Do Register																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																																DO			



Register		Hash_Do		
Bits	Name	Description	Reset Value	Access
31:1	(Reserved)	These bits are always 0.	N/A	WO
0	DO	Hash Do 1: Begin Hash Computation	N/A	WO

26.4.3 Hash Interrupt Register

Register Name:						Hash_Int																													
Block Base Address:						0x7000_						Offset Address						2208						Reset Value						0x00000000					
Register Description:						Hashing Coprocessor Interrupt Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
(Reserved)																																INT			

Register		Hash_Int		
Bits	Name	Description	Reset Value	Access
31:1	(Reserved)	These bits are always 0.	0x0	RW
0	INT	Interrupt • 1: Interrupt active • 0: Interrupt inactive Write 0x1 to clear	0	RW

26.4.4 Hash Chain Register

Register Name:						Hash_Chain																													
Block Base Address:						0x7000_						Offset Address						220C						Reset Value						0x00000000					
Register Description:						Hashing Coprocessor Chain Register																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Hash Chain																																			

Register		Hash_Chain		
Bits	Name	Description	Reset Value	Access
31:0	Hash Chain	Hash Chaining variable.	0x0	RW



26.4.5 Hash Data FIFO

Register Name:						Hash_Data																													
Block Base Address:						0x7000_						Offset Address						2210						Reset Value						0x00000000					
Register Description:						Hashing Coprocessor Data Register																		Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Hash Data																																			

Register		Hash_Data			
Bits	Name	Description	Reset Value	Access	
31:0	Hash Data	Hash Data FIFO.	0x0	RW	

*** NOTE TO EEAS REVIEWERS: this register was tagged “MAS only” but should it be EEAS also? I thought Rockwell needed all the debug info. ***

26.5 Compatibility

The SHA-1 algorithm is compliant to the FIPS 180-1 standard (<http://www.itl.nist.gov/fipspubs/fip180-1.htm>).

The MD5 algorithm is compliant to RFC 1321 (<http://www.faqs.org/rfcs/rfc1321.html>).

§ §





27.0 AHB Queue Manager (AQM)

27.1 Overview

The AHB Queue Manager (AQM) provides queue functionality for various cores. It maintains the queues as circular buffers in an embedded, 8-Kbyte SRAM. It also implements the status flags and pointers required for each queue.

The AQM manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The AQM interfaces include an Advanced High-performance Bus (AHB) interface to the NPEs and Intel XScale processor (or any other AHB bus master), a Flag Bus interface, an event bus (to the NPE condition select logic) and two interrupts to the Intel XScale processor. The AHB interface is used for configuration of the AQM and provides access to queues, queue status and SRAM. Individual queue status for queues 0-31 is communicated to the NPEs via the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs via the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

Read or write entries to a queue, will be accomplished by performing AHB read/write accesses to any of the corresponding Queue Access Register addresses. The AQM will intercept these accesses, since no physical data resides at these addresses, and lookup the appropriate queue pointer to perform the requested read or write. Upon a read or write access to a queue, the pointers and status for the queue are updated as needed. Further detail is given in the following sections.

The lower queues provide individual status to the NPE Condition Coprocessor (CCP) via the flag bus interface. This means that any of these queues can be individually assigned to events and scheduled accordingly. The upper queues have historically been managed as though they were a monolithic resource, where the resource is the entire group of queues and each queue is a sub-instance. Because the status for these upper queues was not directly connected to events, and the status flags are flattened in the status register, a software polling approach was used to determine which of the queues required attention. For this version of the AQM, this division between the upper and lower queues has been somewhat formalized. This definition has the upper queues providing all status, but in a flat fashion across each type of status register and the addition of the ability to schedule the upper queues as if they were a monolithic resource via a programmable event funnel into event bus flags. What these differences mean will be discussed in later sections.

27.2 Feature List

- Provides queue functionality for NPEs and the Intel XScale® Processor
- Manages 64 independent queues
- Implements queues as FIFOs with circular buffer rotation in SRAM
- Implements read/write pointers for each queue in SRAM
- Programmable queue entry size supported (queues may be configured for 1, 2, or 4 word entries)



- Programmable queue size supported (queues may be configured for 16, 32, 64 or 128 word buffer)
- Maintains empty (E), nearly empty (NE), nearly full (NF), and full (F) status flags on each of the queues 0-63
- Programmable queue watermarks for assessing NE and NF queue status flags
- Provides status flag information, E, NE, F and NF, for queues 0-31 to the NPEs via a common Flag Bus
- Provides Underflow and Overflow Status Flags for each of the queues 0-31
- Programmable event status enable for queues 32-63
- Two Intel XScale processor interrupts, one for queues 0-31 and one for queues 32-63
- Individual interrupt enables for each queue
- Programmable interrupt source for each of the queues 0-31 as the assertion or de-assertion of 1 of 4 status flags, E, NE, NF or F
- NE status flag used as the interrupt source for each of the queues 32-63
- Provides read/write access to all queues, queue pointers, status flags, configuration registers, interrupt registers and SRAM via the AHB
- SRAM core computes and checks parity and provides error registers

27.3 Block Diagram

Figure 207. AHB Queue Manager

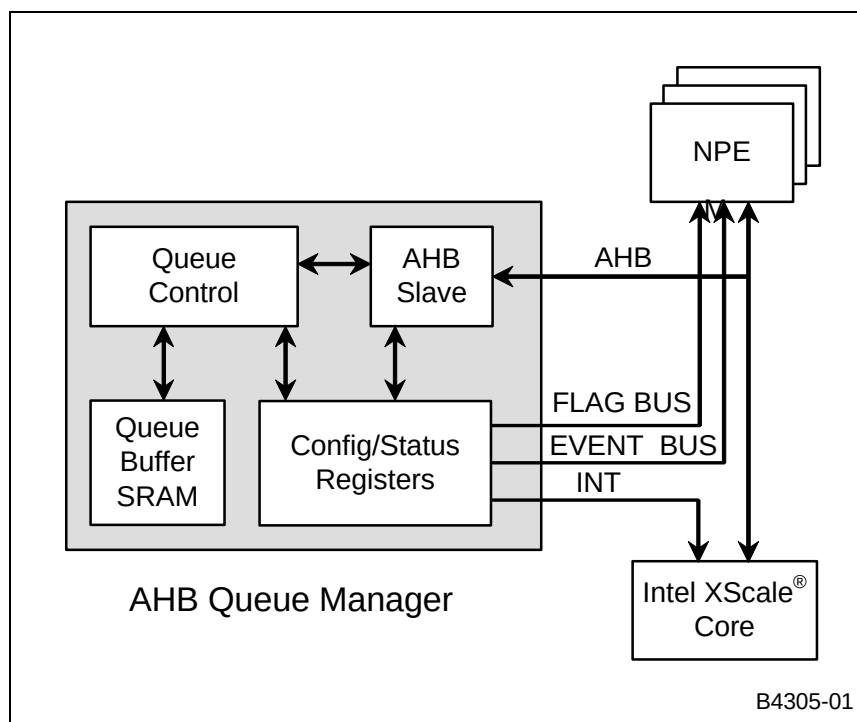


Figure 207 shows a high level block diagram of the AQM and its connections. The flag bus present(s) a unified flag bus to each of the NPE CCPs (Condition Coprocessors) that are to be connected. AQM instantiations are driven from the AHB bus and all operations



are necessarily constrained to occur in the order they occur on the AHB. The timing between an AHB operation and a flag bus update is fixed. Not all NPEs are required to be connected to the flag bus via their CCP, so this connection scheme is very flexible.

The event outputs are also connected to the NPE CCP and wired to the appropriate events as the top level system requires. Each AQM produces three event outputs 'A', 'B' and 'C'.

27.4 AHB Interface

The AHB interface provides read/write access to all AQM configuration/status registers, queues and SRAM. The AQM is a slave with a 32 bit data bus configuration on the AHB. The address map for the AQM is shown in [Table 293](#), the addresses listed in the table are relative offsets from the from the AQM base address. Unsupported exceptions to the AHB slave requirements include accesses with a data transfer size of byte or half-word, wrapping burst accesses and 16 beat incrementing burst accesses. These accesses will result in an AHB Error response. Accesses to any unused locations within the AQM address space will result in an OKAY response on the AHB. Read accesses to the unused address locations will result in zeroes returned on the AHB. The AQM will not perform any internal operations on write accesses to any unused locations. Data formats for all registers accessible via the AHB is given in [Section 27.6, "Register Descriptions"](#) on page 939.

Table 293. AHB Queue Manager Memory Map (Sheet 1 of 2)

Address	AQM Function
0x03FFC	64 Queue Buffer Space - SRAM 1,984 x 4 Bytes
0x02100	
0x020FC	64 Queue Configuration Words - SRAM 64 x 4 Bytes
0x02000	
0x01FFC	UNUSED ADDRESS SPACE
0x00468	
0x00464	Reserved
0x00460	Error Control Register
0x0045C	Queue 0 to 31 Status Map Register
0x00458	Queue 32 to 63 Event Source Register
0x00454	Reserved
0x00450	Queue 32 to 63 Event 'C' Enable Register
0x0044C	Queue 32 to 63 Event 'B' Enable Register
0x00448	Queue 32 to 63 Event 'A' Enable Register
0x00444	Queue 32 to 63 Nearly Full Status Register
0x00440	Queue 32 to 63 Empty Status Register



Table 293. AHB Queue Manager Memory Map (Sheet 2 of 2)

0x0043C	Queue 0 to 63 Interrupt Register 2 x 4 Bytes
0x00438	
0x00434	Queue 0 to 63 Interrupt Enable Register 2 x 4 Bytes
0x00430	
0x0042C	Queue 0 to 31 Interrupt Status Flag Source Select Register 4 x 4 Bytes
0x00420	
0x0041C	Queue 32 to 63 Full Status Register
0x00418	Queue 32 to 63 Nearly Empty Status Register
0x00414	Queue 0 to 31 Underflow/Overflow Status Register 2 x 4 Bytes
0x00410	
0x0040C	Queue 0 to 31 Status Registers 4 x 4 bytes
0x00400	
0x003FC	Queue 0 to 63 Read/Write Access 64 x 16 bytes
0x00000	

27.4.1 Queue Control

The queues are implemented as circular buffers where adding an entry is performed by a write to a queue and removing an entry is performed by a read from a queue. Entries are read from a queue in the same order in which they were written to the queue. The read/write pointers track the removal/addition of entries from/to a queue. The queue control performs the autonomous access of the queues. External agents wanting to access a queue, will perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM will perform the requested access to the queue in SRAM. Support is provided for 64 queues. Upon receiving a queue read or queue write from the AHB interface, queue control fetches the selected queue configuration from SRAM. The queues or circular buffers will reside in internal SRAM. Configuration for each queue will consist of:

- A Base Address — The address where the queue starts and is configurable for placing the queue buffer on any 16 word boundary within the SRAM address range of 000H to 7C0H (word address). These SRAM address ranges correspond to AQM address ranges 0x2000 to 0x3FFC respectively. Note that the lower SRAM addresses are used to store the queue control words and therefore should not be used for queue data storage.
- A Write Pointer — A pointer to the next queue location to be written and is maintained by queue control.
- A Read Pointer — A pointer to the next location to be read and is maintained by queue control.
- Queue Entry Size — The size of each queue entry. This is configurable for 1, 2, or 4 words.
- Queue Size — The number of words allocated to the queue and is configurable for 16, 32, 64, or 128 words.
 - Examples of how to use the Queue Entry Size and Queue Size parameters to obtain number of Q-entries for each of the 64 queues when program a Q-Manager size of 8KB



- Queue Size(128 words)/Queue Entry Size(4 words) = 32 entries of 4 words each
- Queue Size(128 words)/Queue Entry Size(2 words) = 64 entries of 2 words each
- Queue Size(128 words)/Queue Entry Size(1 word) = 128 entries of 1word each
- Queue Size(32 words)/Queue Entry Size(4 words) = 8 entries of 4 words each
- One more note, when the maximum Q-Manager SRAM utilization is 1984 Words (assuming all queues are being used), the Queue size cannot be chosen to be the maximum size for every queue. For example 1984Words/128 words = 15 queues worth of data with 64 words worth of data left over. Note that since one cannot program a queue size of 15, either the next smallest queue depth will need to be used or the queues will need to be different depths.
- NE watermark – this field indicates the maximum number of occupied entries for which a queue is considered to be nearly empty. It can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.
- NF watermark – this field indicates the maximum number of empty entries for which a queue is considered to be nearly full. It can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.



Figure 208. Representative Logical Diagram of a Queue

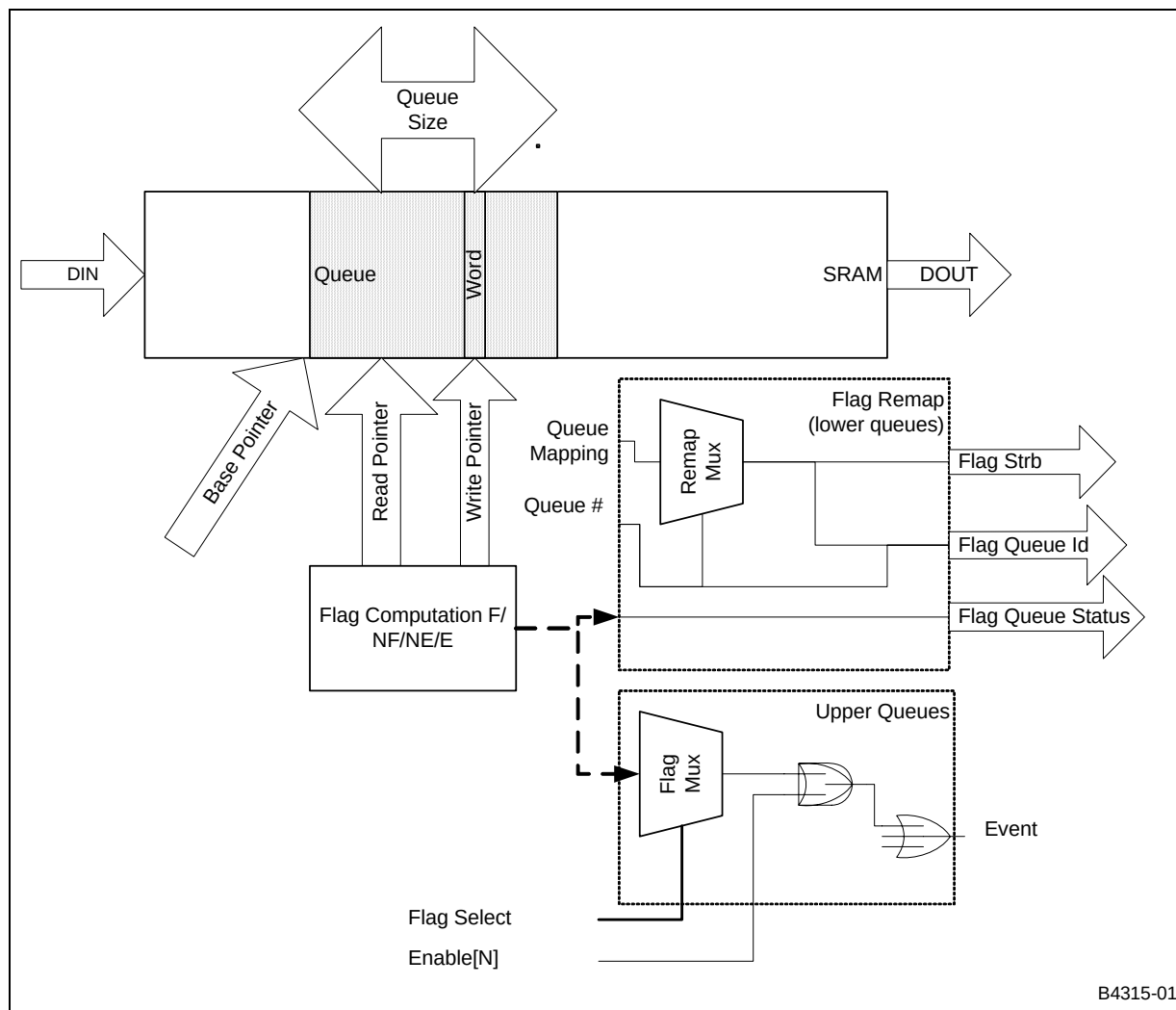


Figure 208 shows the logical diagram of what any given queue looks like. The actual implementation of the AQM is as a monolithic controller of queues, but you can look at each queue as if it stands alone. Each queue fits into the SRAM at a certain base address, is a certain size, and maintains its own flags. The output of those flags depends on whether the queue is a lower queue (0-31) or an upper queue (32-63). The detailed description follows.



To access any given queue, after the selected queue configuration is read from SRAM, the queue base address is summed with the read or write pointer to form the queue address. If the queue isn't empty, when the request is a read, or full, when the request is a write, queue control will perform the requested queue access at the calculated queue address. If the request is a read, the queue data read from SRAM at the calculated queue address is passed to the AHB interface for the corresponding data acknowledge to the AHB read request. If the request is a write, the data from the AHB interface is written into SRAM at the calculated queue address. When the read and write pointers are equal, the queue is either full or empty as determined by the full or empty status flags.

When a read request of an empty queue buffer is performed, queue control will return zeroes in the data field to the AHB interface and will set the Underflow Status Flag. When a write request of a full queue is performed, the data is discarded and queue control will set the Overflow Status Flag. Underflow and Overflow Status Flags are maintained on queues 0-31 only. Otherwise for a read request of an empty queue or write request of a full queue, queue control will not perform any action upon the queue buffer or queue configuration word.

Following the queue access, the appropriate read or write pointer, as indicated by the type of queue access, is incremented and the queue configuration word, with all other fields maintained, is written back into SRAM. The queue size will be used in determining the number of active bits within the allocated 7 bit field for the read and write pointers. Configurable queue sizes of 16, 32, 64 and 128 directly correspond to read and write active bit widths of 4, 5, 6, and 7. The unused bits of the read and write pointers will be zeroed prior to writing the queue configuration word back into SRAM.

One to four Queue Access Register addresses, 0, 4, 8, and 0xC, are allocated per queue as determined by the queue's programmed entry size. Thus a queue, with an entry size set to one word, will support accesses to the first location, 0, and a queue with an entry size set to two, will support accesses to the first and second locations, 0 and 4. A queue with an entry size set to four words will support accesses to all four locations. Accesses to the non-supported locations will not be performed and queue read/write pointers will be unchanged. A queue, with a programmed entry size of two or four words, requires the two or four accesses of each queue entry to be performed sequentially beginning with address 0. Accesses performed out of order will not be performed. Incomplete accesses (e.g. reading only the first two words of a four word entry) will not update the pointers. The sequential accesses can be performed via multiple single word accesses or via a burst access on the AHB. If a Queue burst access of more than four words is attempted, the AQM will perform the accesses to the first two or four locations, as determined by the queue's entry size, and the remainder of the accesses of the burst will not be performed. Queue read accesses which are not performed, will return zeroes in the data field on the AHB.

Note that the entry size parameter does not affect the data type. The AQM does not know the semantic meaning of the entries in the queues, they are all simply 32-bit words. These entries may be pointers, data blocks, big-endian packed data or little-endian packed data. The entry size does not really change anything except how the AHB *can* optimize a transfer. For example, if an entry size is set to two words, a device on the AHB can read and write this queue in a bus burst operation of length two, which will optimize bus access. An entry size of four words enables a bus burst of length 4. Obviously, to support bursting on AHB, one must have consecutive addresses since that is how the protocol works, and so to support this the entries have multiple addresses. Of course individual reads and writes may be performed to these consecutive addresses, but unless there is some kind of bursting, the difference between individual reads to the same location and individual reads to consecutive locations is small. This means that unless a given AHB source device is using a burst access to read or write the AQM there is little AHB performance benefit to programming the word size of two or four (although there may be software advantages).



27.4.2 Queue Status

Status information for the 64 queues is provided in the status registers. Six status flags will be maintained for each of the queues 0-31, empty, nearly empty, full, nearly full, underflow, and overflow. Only the four Empty, Nearly Empty, Nearly Full, and Full Status Flags are provided for queues 32-63. The status flags will be read/write accessible via the AHB, although the arrangement of flags differs between queues 0-31 and 32-63. The Flag Bus will communicate status for queues 0-31 to the NPEs, the status funnel¹ will communicate status for queues 32-63 to the event inputs to the NPEs, and two interrupts will provide status interrupting capability for the Intel XScale processor. The following sections outline the queue status requirements.

27.4.2.1 Status Update

Following any updates for a queue access, the read and write pointers will be used in determination of status flag settings for the accessed queue. When the queue entry size is set to 1, status flags are updated following every queue access. If the queues are set for multi-word entry sizes, 2 or 4, the status flags will be updated following the queue access, that completely fills or empties a queue entry. If the read and write pointers are equal and the last access to the queue was a read, then the queue is empty. If the read and write pointers are equal and the last access was a write, then the queue is full. The nearly empty and nearly full configurable watermarks are used in determining the settings for the NE and NF Status flags. These watermarks can be set to 0, 1, 2, 4, 8, 16, 32, or 64 entries. If the number of completely empty entries is less than or equal to the full watermark, the queue is considered nearly full. If the number of completely full entries is less than or equal to the empty watermark, the queue is considered nearly empty.

Of course, status may be read at any time by an AHB read of the appropriate status register. The status read will reflect the status of the queue at the time of the read, respecting all previous AHB operations. By example, if two closely spaced AHB reads are performed, one to a queue and another to that queue's status register, the status register read will reflect the completion of the queue read. This will require that the second read have additional wait states inserted (by hardware) into the bus operation in order to insure that the first operation has been completed.

The operation of the Nearly Full, Nearly Empty, Full and Empty Status Flags with the Nearly Empty and Nearly Full watermarks set to various levels is demonstrated in [Table 294, "Queue Status Flags" on page 934](#). For this example, the buffer size is set to 64 and the entry size is set to 1. When the watermarks are set to zero, the Nearly Empty and Empty Flags will be identical and the Nearly Full and Full Flags will be identical.

Please note in the following table the "# Entries in the Queue" represents the cardinal number of entries in the queue for each watermark value.

1. A "status funnel," in this context, is the logical OR of a status word across the field. For example, if any bit of the Empty status register is set, the empty status funnel is true.



Table 294. Queue Status Flags

Nearly Empty Watermark	Nearly Full Watermark	# Entries in the Queue	E	NE	NF	F
0 (000)	0 (000)	0 1 – 63 64	1 0 0	1 0 0	0 0 1	0 0 1
1 (001)	1 (001)	0 1 2 – 62 63 64	1 0 0 0 0	1 1 0 0 0	0 0 0 1 1	0 0 0 0 1
2 (010)	2 (010)	0 1 – 2 3 – 61 62 – 63 64	1 0 0 0 0	1 1 0 0 0	0 0 0 1 1	0 0 0 0 1
4 (011)	4 (011)	0 1 – 4 5 – 59 60– 63 64	1 0 0 0 0	1 1 0 0 0	0 0 0 1 1	0 0 0 0 1
8 (100)	8 (100)	0 1 – 8 9 – 55 56 – 63 64	1 0 0 0 0	1 1 0 0 0	0 0 0 1 1	0 0 0 0 1

27.4.2.2 Status Interrupts

Two processor interrupts will be provided, one for queues 0-31 and one for queues 32-63. Each of the interrupt signals is computed as a masked 32-way logical-OR of one edge-sensitive status bit per queue. In other words, each queue contributes a single edge-sensitive input into one of the 32-way logical-OR combinations. For queues 0-31, this input is independently configurable. It may be a positive or negative edge-sensitive version of any one of the E, NE, NF or F status flag bits. The selected status flag may be different for each queue. For queues 32-63, the input is always the NE status flag bit with a positive edge-sensitive version only. The set of selected 32 condition signals is masked by the corresponding interrupt enable register but the value of INT0SRCSELREG0 bit 3 affects how this happens in a subtle way.

There is a bit in INT0SRCSELREG0 which will modify the reset operation of the interrupts. If bit 3 of this register is set to 0, then these interrupts are generated for active high, level triggered usage. On occurrence of the selected transition of one or more of the status flag sources, an active high interrupt level is registered. Via the AHB, the processor can read a 32-bit Interrupt register to determine the source or sources for each interrupt. Selective interrupt reset capability will be provided for each of the queue sources via writing a one to the appropriate queue bit(s) within the interrupt register. Upon clearing (i.e. writing a '1' to) the appropriate bit(s) in the Interrupt Register, the interrupt cannot be generated again by the same source, until the active status flag condition is removed and then are asserted again. If the interrupt is reset, but the condition is still active, the interrupt register will not reflect the status. In this mode, the interrupt enable register affects whether or not the bit may be set, and the interrupt is simply the logical 'OR' of the interrupt register.



If this bit is set to 1, then the interrupts will only reset if the interrupting condition has also been cleared when the write to the QUEINTREG occurs. In other words, this bit determines if the interrupt is globally rising edge sensitive (INT0SCRSELREG0[3] is '0') or is level sensitive (INT0SCRSELREG0[3] is '1'). In addition, the interrupt mask is applied *after* the interrupt register. This means that the QUEINTREG register will reflect the active status of masked interrupts.

Notice that there is a subtle difference in how the interrupt mask is applied in these two cases. If the bit is '0' the interrupt mask prevents the *setting* of the interrupt status register bit whereas if the bit is '1' the interrupt mask prevents the *propagation* of the interrupt status register bit. This introduces a corner case. If the bit is set to a one, an interrupt is generated on a queue and then that interrupt is subsequently disabled, but it is either not cleared (because there is no write to QUEINTREG) or is not clearable (because the condition is still true and in this mode it is level sensitive) then if you change the mode to '0' the interrupt mask is no longer applied, and a (spurious) interrupt will be generated. The interrupt itself cannot be prevented, but after the mode switch the interrupt may be cleared and subsequently will remain masked.

27.4.3 AQM SRAM

The queue buffers and queue configuration words managed by the AQM reside in on-chip SRAM. The SRAM is a 2K by 36 bits wide implementation (where the "extra" 4 bits are used for parity protection). It is a single port synchronous SRAM that supports write accesses and a 133 MHz clock. The SRAM is fully accessible via the AHB interface to the AQM as well as the queue control for queue access. The queue configuration words are located in the bottom 64 words of the SRAM with the remainder of the SRAM being allocated to queue buffers. The SRAM may be freely written and read, although writes will have obvious side effects. Overwriting the configuration words results in unpredictable behavior in the queues. Therefore, reads of the SRAM may not commingle with the normal usage of the queue manager's queues. Reading and writing the SRAM may either be done as part of a diagnostic test or as part of the normal operation of the AQM.

The existence of parity requires that the memory be initialized to a defined pattern before it is read. The SRAM is divided into two regions, the Queue Configuration Words and the buffer space. Reading a location either directly or by operating the AQM in its natural mode without initializing the location is most likely going to produce a parity error. For this reason, software must take care to either initialize the AQM SRAM or guarantee that no reads are ever performed before a write. To initialize the SRAM for a strict queue usage model, all that is necessary is to initialize the queue control word for the specific queue and to guarantee that no reads are done before the first write. Accessing the SRAM for direct access operations (i.e. addresses > 0x2000) may produce corner cases where "left over" pointers (from previous operations) to uninitialized locations cause spurious and hidden reads. For this reason, if the queue manager SRAM is expected to support direct access operations, the entire SRAM will have to be initialized to guarantee that no spurious errors will be detected. The "safest" solution is for the boot software to always initialize the SRAM to a constant value (such as 0x00000000) after power-up or reset. Parity errors will not be reported if parity reporting is disabled in the address error register, and this bit should be left in the disable position until memory is initialized.

A parity error may be signaled via an interrupt to the Intel XScale processor or some other mechanism. When the Error bit in the QUEADDERR register is true the `aqm_parity_error` port is true, and remains true until the Error bit is cleared. There is little or no recourse for parity errors, and the detection of a parity error indicates that a fatal and unrecoverable error has occurred in the AQM SRAM. However, the logical operation of the AQM is marginally intact. Should the parity error occur on a data fetch only the data is corrupted. Should the parity error occur on a configuration word fetch, the entire operation of that queue (and whether it interferes with the operation of another queue) may be compromised. It should be assumed that any parity error



requires essentially a complete software reset of the AQM and in general any stored data is invalid. To accomplish this software reset, the AQM must have parity disabled and the entire memory should be reinitialized. After reinitialization “normal” operation may resume, although obviously any stored data is now destroyed.

27.4.4 Data Validity

The AHB queue manager can produce “incorrect” data under certain contexts, and these contexts need to be understood. Some of these cases represent incorrect software usage of the AQM, others represent hardware conditions. The following table summarizes the cases where the AQM does not return “valid” data, how a software stack will be notified and what (if anything) can be done.

Table 295. Data Validity Cases and Their Handling

Case	Hardware Signal	Software Signal	Recovery
Unsupported bus operation	AHB Error response	Data Abort fault to Intel XScale® Processor Error to AHB COP Target Abort to PCI Forced parity error to expansion bus - may result in silent data corruption if expansion device ignores parity.	None
Out of range SRAM address	None	Zero value returned on all reads, writes ignored.	Detect zero value, handle as exception
Reading incorrect number of queue words	None	Zero value returned on all reads, writes ignored.	Detect zero value, handle as exception
Queue overflow on write	None	Status flag set.	None, data is lost.
Queue underflow on read	None	Status flag set, zero value returned	Detect zero value, handle as exception
Parity error without parity detection enabled	None	None, will most probably result in silent corrupted data.	None
Parity error on data during queue read	AHB Error response	Data Abort fault to Intel XScale® Processor Error to AHB COP Target Abort to PCI Forced parity error to expansion bus - may result in silent data corruption if expansion device ignores parity. Interrupt signal to Intel XScale processor	Reset and restart application usage of AQM

As described elsewhere, not all AHB cycle types are supported, with certain ones automatically producing an Error response. The data returned for these cases is not defined.

Data returned for out-of-range SRAM access or MMR access is always zero and no error condition is signaled. If software should attempt reads or writes of out-of-range addresses, there will not be any obvious notification except that the data read will be zero and data written will be ignored.

Reading (for example) two words out of a single word queue will return data for the first word and zeros for the second word. The fact that the second word is not defined is not signaled. In general, a mismatch between the number of words in the entry between the actual usage and the configuration word definition will either result in lost data (i.e. as if it were overflow) or zeros.



On the overflow condition, the written data is permanently lost. On the underflow condition, the data returned is zero.

If there are parity errors where the parity notification is not enabled, the data returned represents the data containing the parity error. If parity notification is enabled and the entry contains a parity error, the error is signaled on that particular bus cycle and consistent with the AHB specification, the master has the option of terminating a burst operation on the first error.

In general, a software usage model should treat the return of zero data as a suspicious case and the data type stored in the AQM should take this into account. If the data type were to be memory address pointers, this is well behaved since a null pointer is not defined. Other data types should take this into account.

27.4.5 Burst Operations to Queues

Burst operations are useful for writing multiple entry queues. An INCR4 burst can fill an entire entry for a four entry queue without having to re-arbitrate for ownership of the AHB bus, which enhances performance. However, if the burst type does not match the entry size for the queue, the transaction data to the inactive queue entries is simply dropped. In general, software should avoid burst transactions to queues that have fewer words per entry than the burst because of the decreased performance.

27.5 Detailed Register Descriptions

Table 296. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

Table 297. Register Summary (Sheet 1 of 2)

Address	Register Name	Description	Reset Value	Access
0x60000000	QUEACC0_0	Queue 0 word 0 data register	N/A	RW
0x60000004	QUEACC0_1	Queue 0 word 1 data register (used only when Queue 0 has a entry size of 2 or 4)	N/A	RW
0x60000008	QUEACC0_2	Queue 0 word 2 data register (used only when Queue 0 has a entry size of 4)	N/A	RW
0x6000000C	QUEACC0_3	Queue 0 word 3 data register (used only when Queue 0 has a entry size of 4)	N/A	RW
0x60000010	QUEACC1_0	Queue 1 word 0 data register	N/A	RW
0x60000014	QUEACC1_1	Queue 1 word 1 data register (used only when Queue 1 has a entry size of 2 or 4)	N/A	RW
0x60000018	QUEACC1_2	Queue 1 word 2 data register (used only when Queue 1 has a entry size of 4)	N/A	RW
0x6000001C	QUEACC1_3	Queue 1 word 3 data register (used only when Queue 1 has a entry size of 4)	N/A	RW



Table 297. Register Summary (Sheet 2 of 2)

Address	Register Name	Description	Reset Value	Access
...	...	...	N/A	RW
0x600003F0	QUEACC63_0	Queue 63 word 0 data register	N/A	RW
0x600003F4	QUEACC63_1	Queue 63 word 1 data register (used only when Queue 63 has a entry size of 2 or 4)	N/A	RW
0x600003F8	QUEACC63_2	Queue 63 word 2 data register (used only when Queue 63 has a entry size of 4)	N/A	RW
0x600003FC	QUEACC63_3	Queue 63 word 3 data register (used only when Queue 63 has a entry size of 4)	N/A	RW
0x60000400	QUELOSTAT0	Queue 0-7 complete status	0x33333333	RW
0x60000404	QUELOSTAT1	Queue 8-15 complete status	0x33333333	RW
0x60000408	QUELOSTAT2	Queue 16-23 complete status	0x33333333	RW
0x6000040C	QUELOSTAT3	Queue 24-31 complete status	0x33333333	RW
0x60000410	QUEUOSTAT0	Queue 0-15 Overflow/Underflow Status	0x00000000	RW
0x60000414	QUEUOSTAT1	Queue 16-31 Overflow/Underflow Status	0x00000000	RW
0x60000418	QUEUPSTATNE	Queue 32-63 Nearly Empty Status Flags	0x00000000	RW
0x6000041C	QUEUPSTATF	Queue 32-63 Full Status Flags	0x00000000	RW
0x60000420	INTROSRCSELREG0	Interrupt 0 Status Flag Source Select for Queues 0-7	0x00000000	RW
0x60000424	INTROSRCSELREG1	Interrupt 0 Status Flag Source Select for Queues 8-15	0x00000000	RW
0x60000428	INTROSRCSELREG2	Interrupt 0 Status Flag Source Select for Queues 16-23	0x00000000	RW
0x6000042C	INTROSRCSELREG3	Interrupt 0 Status Flag Source Select for Queues 24-31	0x00000000	RW
0x60000430	QUEIEREG0	Queue Interrupt Enable for Queues 0-31	0x00000000	RW
0x60000434	QUEIEREG1	Queue Interrupt Enable for Queues 32-63	0x00000000	RW
0x60000438	QUEINTREG0	Queue Interrupt for Queues 0-31	0x00000000	RW
0x6000043C	QUEINTREG1	Queue Interrupt for Queues 32-63	0x00000000	RW
0x60000440	QUEUPSTATE	Queue 32-63 Empty Status Flags	0x00000000	RW
0x60000444	QUEUPSTATNF	Queue 32-63 Nearly Full Status Flags	0x00000000	RW
0x60000448	QUEUPEVA	Queue 32-63 Empty Event 'A' Enable	0x00000000	RW
0x6000044C	QUEUPEVB	Queue 32-63 Empty Event 'B' Enable	0x00000000	RW
0x60000450	QUEUPEVC	Queue 32-63 Empty Event 'C' Enable	0x00000000	RW
0x60000454		(Reserved)		
0x60000458	QUEUPSTATSRC	Queue 32-63 Event Source Register	0x00000000	
0x6000045C	QUELOSTATMAP	Flag bus mapping for queues 0-31		RW
0x60000460	QUEADDERR	Parity Error Address and Control	0x00010000	RW
0x60000464		(Reserved)		
0x60002000	QUECONFIG0	Queue 0 configuration	U	RW
...	...	...	U	RW
0x600020FC	QUECONFIG63	Queue 63 configuration	U	RW
0x60002100		SRAM queue data buffer	U	RW
...		SRAM queue data buffer	U	RW
0x60003FFC		SRAM queue data buffer	U	RW



27.6 Register Descriptions

27.6.1 Queue Access Word Registers 0 - 63

External agents wanting to access a queue, will perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM will perform the requested access to the queue in SRAM. See [Section 27.4.1, "Queue Control" on page 929](#) for clarification on AHB queue accesses to the AQM. As described above, these queue access registers are defined in a block of four 32-bit words, where only the first 32-bit word is defined for a word size of one, only the first two 32-bit words are defined for a word size of two and all four 32-bit words are defined for a word size of four.

Register Name:						QUEACC (0 ≤ n ≤ 63)																													
Block Base Address (BBA):						Queue #n 0x0000						Offset Address						BBA + 16n + 4x						Reset Value						Not Applicable					
Register Description:						Queue #n access register. There are 1-4 addresses (0 ≤ x ≤ 3), as determined by the programmed entry size, for requesting read/write accesses to individual queues. No physical data resides at these addresses.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Queue Read/Write Data																																			

Register		QUEACC (0 ≤ n ≤ 63)															
Bits	Name	Description														Reset Value	Access
31:0	Queue Read/Write Data	Queue data word. Addresses addressed consecutively program in multiple entry sizes. For example, QUEACC0 word 0 is at address offset 0x0000 and QUEACC0 word 1 is at address offset 0x0004.														N/A	RW

27.6.2 Queues 0-31 Status Register 0 - 3

The access to these status registers is read/write, however except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually *change* the status, it only writes the shadow register which contains the status.

Register Name:						QUELOWSTAT (0 ≤ n ≤ 3)																																									
Block Base Address:						Reg #n 0x0400						Offset Address						+ 4n						Reset Value						0x33333333																	
Register Description:						Queue status register for the queues 0-31. F/NF/NE/E: '1' – active flag.																		Access:						(See below.)																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
Queue(8n+7)						Queue(8n+6)						Queue(8n+5)						Queue(8n+4)						Queue(8n+3)						Queue(8n+2)						Queue(8n+1)						Queue(8n)					



Register		QUELOWSTAT (0 ≤ n ≤ 3)		
Bits	Name	Description	Reset Value	Access
4k+3 : 4k	Queue(8n+k) Status Flags	(0 ≤ k ≤ 7) Queue (8n+k) complete status flags of: Full / Nearly Full / Nearly Empty / Empty For each flag, '1' is active.	0x3	RW

27.6.3 Underflow / Overflow Status Register 0 - 1

The access to these status registers is read/write, however except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:						QUEUOSTAT (0 <= n <=1)																																																																																									
Block Base Address:						Reg #n 0x0410						Offset Address						+ 4n						Reset Value						0x00000000																																																																	
Register Description:						Queue underflow/overflow status register for the queues 0-31. OF/UF: '1' – Overflow/Underflow has occurred.																Access:						(See below.)																																																																			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																
Queue (16n + 15)						Queue (16n + 14)						Queue (16n + 13)						Queue (16n + 12)						Queue (16n + 11)						Queue (16n + 10)						Queue (16n + 9)						Queue (16n + 8)						Queue (16n + 7)						Queue (16n + 6)						Queue (16n + 5)						Queue (16n + 4)						Queue (16n + 3)						Queue (16n + 2)						Queue (16n + 1)						Queue (16n)					

Register		QUEUOSTAT (0 ≤ n ≤ 1)															
Bits	Name	Description														Reset Value	Access
2k+1 : 2k	OF/UF	(0 ≤ k ≤ 7) Queue (16n+k) Overflow and Underflow bit, respectively.														'b00	RW

27.6.4 Queues 32-63 Empty Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:						QUEUPSTATE																									
Block Base Address:		0x0440					Offset Address					+ 4n					Reset Value					0xFFFFFFFF									
Register Description:						Queue status register for queues 32-63. E: '1' – flag set.																Access:			(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q63 E	Q62 E	Q61 E	Q60 E	Q59 E	Q58 E	Q57 E	Q56 E	Q55 E	Q54 E	Q53 E	Q52 E	Q51 E	Q50 E	Q49 E	Q48 E	Q47 E	Q46 E	Q45 E	Q44 E	Q43 E	Q42 E	Q41 E	Q40 E	Q39 E	Q38 E	Q37 E	Q36 E	Q35 E	Q34 E	Q33 E	Q32 E



Register		QUEUPSTATE		
Bits	Name	Description	Reset Value	Access
k	Empty	(0 ≤ k ≤ 31) Queue (k) Empty Status Flag.	1	RW

27.6.5 Queues 32-63 Nearly Empty Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:						QUEUPSTATNE																															
Block Base Address:						0x0418						Offset Address						+ 4n						Reset Value						0xFFFFFFFF							
Register Description:						Queue status register for queues 32-63. NE: '1' – flag set.																				Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
Q63 NE	Q62 NE	Q61 NE	Q60 NE	Q59 NE	Q58 NE	Q57 NE	Q56 NE	Q55 NE	Q54 NE	Q53 NE	Q52 NE	Q51 NE	Q50 NE	Q49 NE	Q48 NE	Q47 NE	Q46 NE	Q45 NE	Q44 NE	Q43 NE	Q42 NE	Q41 NE	Q40 NE	Q39 NE	Q38 NE	Q37 NE	Q36 NE	Q35 NE	Q34 NE	Q33 NE	Q32 NE						

Register		QUEUPSTATNE															
Bits	Name	Description														Reset Value	Access
k	Nearly Empty	(0 ≤ k ≤ 31) Queue (k) Nearly Empty Status Flag.														1	RW



27.6.6 Queues 32-63 Nearly Full Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:					QUEUPSTATNF																																
Block Base Address:		0x0444					Offset Address					+ 4n					Reset Value					0x00000000															
Register Description:						Queue status register for queues 32-63. NE: '1' – flag set.																				Access:				(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
Q63 NF	Q62 NF	Q61 NF	Q60 NF	Q59 NF	Q58 NF	Q57 NF	Q56 NF	Q55 NF	Q54 NF	Q53 NF	Q52 NF	Q51 NF	Q50 NF	Q49 NF	Q48 NF	Q47 NF	Q46 NF	Q45 NF	Q44 NF	Q43 NF	Q42 NF	Q41 NF	Q40 NF	Q39 NF	Q38 NF	Q37 NF	Q36 NF	Q35 NF	Q34 NF	Q33 NF	Q32 NF						

Register		QUEUPSTATNF			
Bits	Name	Description	Reset Value	Access	
k	Nearly Full	(0 <= k <= 31) Queue (k) Nearly Full Status Flag.	0	RW	

27.6.7 Queues 32-63 Full Status Register

The access to these status registers is read/write, however except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register which contains the status.

Register Name:						QUEUPSTATF																													
Block Base Address:						0x041C						Offset Address						+ 4n						Reset Value						0x00000000					
Register Description:						Queue status register for queues 32-63. F: '1' – flag set.																		Access:						(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Q63 F	Q62 F	Q61 F	Q60 F	Q59 F	Q58 F	Q57 F	Q56 F	Q55 F	Q54 F	Q53 F	Q52 F	Q51 F	Q50 F	Q49 F	Q48 F	Q47 F	Q46 F	Q45 F	Q44 F	Q43 F	Q42 F	Q41 F	Q40 F	Q39 F	Q38 F	Q37 F	Q36 F	Q35 F	Q34 F	Q33 F	Q32 F				

Register		QUEUPSTATF			
Bits	Name	Description	Reset Value	Access	
k	Nearly Full	(0 <= k <= 31) Queue (k) Nearly Full Status Flag.	0	RW	

27.6.8 Interrupt 0 Status Flag Source Select Register 0 – 3

The interrupt source for each queue is selectable as the positive or negative (NOT) edge-sensitive version of any one of the E, NE, NF or F status flag bits on interrupt 0, aqm_int[0]. The selection is configurable for interrupt 0 only. Interrupt 1, aqm_int[1], is hard wired to the NE Status Flag bit.



Register Name:					INT0SRCSELREG (0 ≤ n ≤ 3)																																		
Block Base Address:					Reg #n 0x0420					Offset Address					+ 4n					Reset Value					0x00000000														
Register Description:					Status Flag selection for interrupt 0 source on queues 0-31.															Access:					(See below.)														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
Resv'd	Queue (8n + 7) Stat Src Sel				Resv'd	Queue (8n + 6) Stat Src Sel				Resv'd	Queue (8n + 5) Stat Src Sel				Resv'd	Queue (8n + 4) Stat Src Sel				Resv'd	Queue (8n + 3) Stat Src Sel				Resv'd	Queue (8n + 2) Stat Src Sel				Resv'd	Queue (8n + 1) Stat Src Sel				Spec for 0	Queue (8n) Stat Src Sel			

Register		INT0SRCSELREG (0 <= n <= 3)		
Bits	Name	Description	Reset Value	Access
4k+2 :4k	Status Source Select	(0 <= k <= 7) 'b000: Queue (8n+k) Empty going TRUE 'b001: Queue (8n+k) Nearly Empty going TRUE 'b010: Queue (8n+k) Nearly Full going TRUE 'b011: Queue (8n+k) Full going TRUE 'b100: Queue (8n+k) Empty going FALSE 'b101: Queue (8n+k) Nearly Empty going FALSE 'b110: Queue (8n+k) Nearly Full going FALSE 'b111: Queue (8n+k) Full going FALSE	000	RW
3	Clear Interrupt	For INT0SRCSELREG0 (only), bit 3 is a configuration for the interrupt operation. If bit 3 is a 0, its reset value, an interrupt bit will clear when a 1 is written back to it in the QUEUEINTREG., even if the interrupting condition is still true. If bit 3 is a 1, then interrupt bits will clear when a 1 is written to QUEUEINTREG <i>and</i> the interrupting condition has been satisfied. All 64 interrupts behave according to this bit. The impact of this bit is shown in the following pseudo-code: if (INT0SRCSELREG0[3] == FALSE) then QUEUEINTOREG bits are set when the selected state change happens AND the bit is enabled in the QUEIEREG register QUEUEINTOREG bits are reset when the bit is written with a 1 AQM_INT is true if any QUEUEINTOREG bits are true else QUEUEINTOREG bits are set when the selected state change happens QUEUEINTOREG bits are reset when the bit is written with a 1 AND the selected state is no longer true AQM_INT is true if any (QUEIEREG AND QUEUEINTOREG) bits are true endif However, in one corner case, if INT0SRCSELREG0[3] == TRUE and QUEUEINTOREG bits get set, if you then set INT0SRCSELREG0[3] == FALSE, the interrupt is generated even if the QUEIEREG bit corresponding to a QUEUEINTOREG bit is FALSE. This occurs because of the difference in the AQM_INT between the two cases.	0	RW



27.6.9 Queue Interrupt Enable Register 0 – 1

Register Name:						QUEIEREG(0 ≤ n ≤ 1)																									
Block Base Address:		0x0430						Offset Address						+ 4n						Reset Value				0x00000000							
Register Description:						Interrupt enables for the queues 0-63. IE: '1' – Enable.																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q(32n + 31) IE	Q(32n + 30) IE	Q(32n + 29) IE	Q(32n + 28) IE	Q(32n + 27) IE	Q(32n + 26) IE	Q(32n + 25) IE	Q(32n + 24) IE	Q(32n + 23) IE	Q(32n + 22) IE	Q(32n + 21) IE	Q(32n + 20) IE	Q(32n + 19) IE	Q(32n + 18) IE	Q(32n + 17) IE	Q(32n + 16) IE	Q(32n + 15) IE	Q(32n + 14) IE	Q(32n + 13) IE	Q(32n + 12) IE	Q(32n + 11) IE	Q(32n + 10) IE	Q(32n + 9) IE	Q(32n + 8) IE	Q(32n + 7) IE	Q(32n + 6) IE	Q(32n + 5) IE	Q(32n + 4) IE	Q(32n + 3) IE	Q(32n + 2) IE	Q(32n + 1) IE	Q(32n) IE

Register		QUEIEREG(0 <= n <=1)			
Bits	Name	Description	Reset Value	Access	
k	Interrupt Enable	(0 <= k <= 31) Queue (32n+k) Interrupt Enable.	0	RW	

27.6.10 Queue Interrupt Register 0 – 1

Two interrupt registers correspond to the two AQM interrupts, `aqm_int[0]` and `aqm_int[1]`. Queue Interrupt register 0 represents queues 0-31, while register 1 represents queues 32-63. Following an interrupt, the appropriate register can be read to determine which queues caused the interrupt. To clear any bit in the interrupt register, write a 1 to the appropriate bit position. Writing a 1 to a bit in the Interrupt register provides a reset-only operation for that bit. Clearing all set bits (by writing 1's in those locations) in the Interrupt Register removes the interrupt (de-assert). The interrupt cannot be generated again by the same source, until the active status flag condition is removed and then reasserted again. The `INT0SRCSELREG0` register changes the operation of the interrupt from a rising-edge sensitive operation to a level sensitive operation.

Register Name:						QUEINTREG(0 <= n <= 1)																													
Block Base Address:						0x0438						Offset Address						+ 4n						Reset Value						0x00000000					
Register Description:						Interrupt Register for the 64 queues. INT: '1' – interrupt occurred.																		Access:			(See below.)								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Q(32n + 31) INT	Q(32n + 30) INT	Q(32n + 29) INT	Q(32n + 28) INT	Q(32n + 27) INT	Q(32n + 26) INT	Q(32n + 25) INT	Q(32n + 24) INT	Q(32n + 23) INT	Q(32n + 22) INT	Q(32n + 21) INT	Q(32n + 20) INT	Q(32n + 19) INT	Q(32n + 18) INT	Q(32n + 17) INT	Q(32n + 16) INT	Q(32n + 15) INT	Q(32n + 14) INT	Q(32n + 13) INT	Q(32n + 12) INT	Q(32n + 11) INT	Q(32n + 10) INT	Q(32n + 9) INT	Q(32n + 8) INT	Q(32n + 7) INT	Q(32n + 6) INT	Q(32n + 5) INT	Q(32n + 4) INT	Q(32n + 3) INT	Q(32n + 2) INT	Q(32n + 1) INT	Q(32n) INT				

Register		QUEINTREG(0 ≤ n ≤ 1)		
Bits	Name	Description	Reset Value	Access
k	Interrupt	(0 ≤ k ≤ 31) Queue (32n+k) Interrupt.	0	RW



27.6.11 Queue Configuration Words 0 - 63

The 64 queue configuration words are located in internal SRAM and require initialization before AQM usage. The read and write pointers need to be cleared on initialization, because this reflects an empty queue. A system reset sets the status registers to reflect empty queues, but until the queue configuration words have been set, this state is somewhat inconsistent. On a write accesses to any of the Queue Configuration Words 0-31, the corresponding queue status is communicated on the Flag Bus. When the AQM is fully configured, use these configuration words for read-only purposes, to monitor queue pointers. Resetting a queue requires two operations. First the appropriate queue status flags must be configured, then the Queue Configuration Word must be set. The data format for the Queue Configuration Word is shown in the QUECONFIG register.

If the queue configuration word has not been initialized, reading or writing to a queue produces undefined operations, and can cause spurious parity errors and/or data corruption of another queue.

Register Name:		QUECONFIG (0 ≤ n ≤ 63)																													
Block Base Address:		Queue #n 0x2000				Offset Address				+ 4n				Reset Value				0xUUUUUUUU													
Register Description:		Queue #n configuration word located in SRAM.																				Access:		(See below.)							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Queue (n) NF Watermark			Queue (n) NE Watermark			Q (n) Buffer Size		Q (n) Entry Size		Queue (n) Base Address						Queue (n) Read Pointer						Queue (n) Write Pointer									

Register		QUECONFIG (0 ≤ n ≤ 63)															
Bits	Name	Description														Reset Value	Access
31:2 9	Nearly Full Watermark	These bits configure the Nearly Full Watermark, to 0 ("000"), 1 ("001"), 2 ("010"), 04 ("011"), 8 ("100"), 16 ("101"), 32 ("110"), or 64 ("111") entries from the top of the queue. The usable range of Nearly Full Watermark selection must be less than the buffer size.														U	RW
28:2 6	Nearly Empty Watermark	These bits configure the Nearly Empty Watermark, to 0 ("000"), 1 ("001"), 2 ("010"), 04 ("011"), 8 ("100"), 16 ("101"), 32 ("110"), or 64 ("111") entries from the bottom of the queue. The usable range of Nearly Empty Watermark selections must be less than the buffer size.														U	RW
25:2 4	Buffer Size	These bits configure the queue buffer size. The buffer size can be configured for 16 ("00"), 32 ("01"), 64 ("10") or 128 ("11") words.														U	RW
23:2 2	Entry Size	These bits configure the queue entry size of the queue. The entry size can be set at 1 ("00") or 2 ("01") or 4 ("10") words. An input of "11" sets the entry size to 1.														U	RW
21:1 4	Base Address	This field configures the starting base address of the queue. The read and write pointer are offset addresses from the base address. This base address is a SRAM 16 word address. (In other words, this value needs to be multiplied by 16 to get the SRAM address. Also, this value is a word address, not a byte address as the queue has no notion of bytes.) Base addresses, 00 to 03, are reserved for the Queue Configuration Words. The most significant bit of the base address is reserved for growth to a 16 KB SRAM to provide 8 KB of additional queue buffer space. Bit 21 is implemented (i.e. can be read and written) but will cause an address wrap (e.g. 0x00 and 0x80 point to the same region in SRAM).														U	RW
13:7	Read Pointer	Pointer to the next entry to read from the queue. The pointer is the AQM's internal SRAM word address. This pointer should be initialized to zero, and should be written only for diagnostic or test purposes.														U	RW
6:0	Write Pointer	Pointer to the next entry to write to the queue. The pointer is the AQM's internal SRAM word address. This pointer should be initialized to zero, and should be written only for diagnostic or test purposes.														U	RW



27.6.12 Queue 32 to 63 Event 'A' Enable Register

This register contains a bit map which enables the chosen class of events (via the Event Source Select Register) through the Event 'A' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'A' event.

Register Name:						QUEUPPEVA																									
Block Base Address:		0x0448					Offset Address					+ 4n					Reset Value					0x00000000									
Register Description:						Queue Event output 'A' enable register.																	Access:			(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register		QUEUPPEVA			
Bits	Name	Description	Reset Value	Access	
k	Enable Empty 'A'	(0 <= k <= 31) Queue (32+k) Event Enable for 'A' Event.	0	RW	

27.6.13 Queue 32 to 63 Event 'B' Enable Register

This register contains a bit map which enables the chosen class of events (via the Event Source Select Register) through the Event 'B' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'B' event.

Register Name:						QUEUPPEVB																									
Block Base Address:		0x044C					Offset Address					+ 4n					Reset Value					0x00000000									
Register Description:						Queue Event output 'B' enable register.																	Access:			(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register		QUEUPPEVB			
Bits	Name	Description	Reset Value	Access	
k	Enable Empty 'B'	(0 <= k <= 31) Queue (32+k) Event Enable for 'B' Event.	0	RW	

27.6.14 Queue 32 to 63 Event 'C' Enable Register

This register contains a bit map which enables the chosen class of events (via the Event Source Select Register) through the Event 'C' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'C' event.



Register Name:						QUEUPPEVC																									
Block Base Address:			0x0450					Offset Address					+ 4n					Reset Value				0x00000000									
Register Description:						Queue Event output 'C' enable register.																Access:			(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register		QUEUPPEVC															
Bits	Name	Description														Reset Value	Access
k	Enable Empty 'C'	(0 ≤ k ≤ 31) Queue (32+k) Event Enable for 'C' event.														0	RW

27.6.15 Event Source Select

For each event, this register allows one to select which class of flags applies to which event. The chosen flags are ANDed with the event enables to form the actual event.

Register Name:						QUEUPSOUSEL																									
Block Base Address:		0x0458						Offset Address						+ 4n						Reset Value				0x000							
Register Description:						Event Source Select Register.																Access:				(See below.)					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)																								Event C Source Select		Event B Source Select		Event A Source Select			

Register		QUEUPSOUSEL															
Bits	Name	Description														Reset Value	Access
5:4	Event Source Select C	Event type source select for event 'C' 'b00 => Empty 'b01 => Nearly Empty 'b10 => Nearly Full 'b11 => Full														'b01	RW
3:2	Source Select B	Event type source select for event 'B'														'b01	RW
1:0	Source Select A	Event type source select for event 'A'														'b01	RW



27.6.16 Queue 0 to 31 Status Selection Map Register

Register Name:						QUELOSTATMAP																													
Block Base Address:						0x045C				Offset Address				+ 4n				Reset Value				Depends on AQM instantiation													
Register Description:						Queue Status Selection register.																Access:				(See below.)									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Resv'd	NPE Sel				Flag Bus Map for Queues 24-31				Resv'd	NPE Sel				Flag Bus Map for Queues 16-23				Resv'd	NPE Sel				Flag Bus Map for Queues 8-15				Resv'd	NPE Sel				Flag Bus Map for Queues 0-7			

Register		QUEUPSTAT0															
Bits	Name	Description														Reset Value	Access
30:2 7	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP will receive this bank at that slot, and if no bits are active, no CCP will receive this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 24-31.														See Note	RW
26:2 4	Map for Queues 24-31	The three bits program in which of the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 24-31, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 26:24 are copied to aqm_flag_id[5:3].														'o3	RW
22:1 9	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP will receive this bank at that slot, and if no bits are active, no CCP will receive this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 16-23														See Note	RW
18:1 6	Map for Queues 16-23	The three bits program in which of the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 16-23, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 18:16 are copied to aqm_flag_id[5:3].														'o2	RW
14:1 1	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP will receive this bank at that slot, and if no bits are active, no CCP will receive this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 8-15														See Note	RW
10:8	Map for Queues 8-15	The three bits program in which of the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 8-15, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 10:8 are copied to aqm_flag_id[5:3].														'o1	RW
6:3	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP will receive this bank at that slot, and if no bits are active, no CCP will receive this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_lag_strb[3:0] for queues 0-7.														See Note	RW
2:0	Map for Queues 0-7	The three bits program in which of the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 0-7, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 2:0 are copied to aqm_flag_id[5:3].														'o0	RW



27.6.17 Queue SRAM Error Data Register

Register Name:					QUEDATAERR																										
Block Base Address:					0x0464					Offset Address					+ 4n					Reset Value					Not Applicable						
Register Description:					Queue SRAM Parity Error Data Register.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data Read																															

Register		QUEDATAERR															
Bits	Name	Description														Reset Value	Access
31:0	Data Read	The value of the data read from the SRAM when the parity error occurred														U	RO

27.6.18 Queue SRAM Error Address/Control Register

The only field to be implemented in this register is the error 'E' field. The other fields are reserved.

Register Name:					QUEADDERR																										
Block Base Address:					0x0460					Offset Address					+ 4n					Reset Value					0x00000000						
Register Description:					Queue SRAM Parity Error Address and Control Register.															Access:					(See below.)						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Reserved)															E	(Reserved)															

Register		QUEADDERR															
Bits	Name	Description														Reset Value	Access
31:17	(Reserved)															0	
16	Error	When a parity error occurs, the AQM will assert its parity output, capture the operation which caused the parity error and set this bit. Writing this bit to a zero via software clears the condition. The value of the address, parity and data fields are stable so long as this bit remains set. This bit is connected to the aqm_parity_error port.														0	RW
15:0	(Reserved)															0	

27.7 Error / Abnormal Conditions

Parity errors are abnormal once the queues have been initialized, and are fatal. Parity errors may come about because of uninitialized operation, or soft errors. Errors due to uninitialized operation are due to an incorrect software initialization flow. Soft errors are an unavoidable consequence of cosmic rays and background radiation and are extremely infrequent, however a single event upset is fatal to the temporary operation of the AQM and is not recoverable in the general case.

§ §





28.0 Error Handling

This section details the types of system level error conditions that may occur in the Intel® IXP45X and Intel® IXP46X Product Line of Network Processors. It also describes the effect of each of these errors.

28.1 Errors

28.1.1 Sources of AHB Bus Errors

AHB Bus errors can occur as a result of a variety of problems. The problems can generally be categorized as either:

- Attempts to access memory locations which are either unimplemented or reserved,
- Attempts to access units with types of accesses the units are not designed to accept (such as INCR 16 for many IXP45X/IXP46X network processors), or
- Data Corruption in the target, indicated by either parity errors or uncorrectable ECC errors in memory.

28.1.1.1 Accesses to Reserved or Unimplemented Addresses

Accesses to reserved or unimplemented memory will result in an AHB error response being asserted by the bus arbiter.

28.1.1.2 Illegal-Access Types

Any unit which receives an access type which it can't respond properly to will respond with an AHB error response. See individual unit descriptions to see what types of transactions each unit is capable of responding to.

28.1.1.3 Expansion Bus Parity Error

When the Expansion Bus Controller receives a parity error on the Expansion Bus, it responds on the AHB bus with an error response to the appropriate master. In addition, it will send an interrupt signal to the interrupt controller, and log the address of the access which faulted.

28.1.1.4 AQM-Parity Error

When the AQM receives a parity error from its internal memory, it will return AHB error to the requesting master. In addition, it will send an interrupt signal to the interrupt controller. As a result of this operation, the operating system must assume the entire queue is corrupt, and it should reset the unit, clearing all data entries in the queue.

28.1.1.5 Memory Controller Unit (MCU), Multiple-Bit, ECC Error

When the MCU unit receives a multiple-bit ECC error, it will return AHB error to the requesting AHB master. In addition, it will send an interrupt signal to the interrupt controller, and log the address of the access which failed.



28.2 Responses to Errors

Most memories internal or external to the IXP45X/IXP46X network processors have some variety of data corruption detection capability. The only significant exception to this rule is the buffer memory of the USB device controller. The Expansion Bus, AQM, and MCU (a.k.a DDRI SDRAM Controller) will all send interrupt signals directly to the interrupt controller indicating data corruption detected by those units. The NPEs will halt as a result of internal data corruption, which will in turn send interrupt signals.

The interrupt unit has a special mechanism which will allow software to set these sorts of interruptions as highest priority interruptions.

28.2.1 PCI Responses to Errors

The PCI Unit will interpret AHB Error as AHB Retry. As a result, it will most likely retry the errored transaction. In the event that the error was caused by a one-time transient condition, this retry will receive good data the second time, and continue operating normally. In the event that this is a continuing system issue, the PCI unit will not receive a valid data response, and will continue this retry until either the condition is repaired or the PCI unit is reset.

In the event of such an error, although the PCI responds as if a retry had occurred, the unit which asserted AHB error will be sending an interrupt to the Interrupt Controller. This allows the Intel XScale processor to know that an error condition exists, and attempt to solve in whatever manner determined appropriate for the given system.

External PCI errors will not be reflected back on the AHB bus.

28.2.2 NPE Responses to Errors

There are three types of NPE soft errors

- NPE IMEM Parity error
- NPE DMEM Parity error
- NPE Coprocessor errors

Multiple concurrent error handling and recovery is not supported in the IXP45X/IXP46X network processors. There are two types of NPE Coprocessor errors:

- AHB error
- Switching Coprocessor error

AHB error is valid on all the NPEs. The causes of AHB error include:

- AHB Slave access error
- Read/Write to an invalid AHB address access
- An illegal transfer size access on the AHB Bus.

The AHB Slave access error can be caused by one of four sources:

- an uncorrectable ECC error on DDRI read access
- parity error on a Queue Manager read Access
- a parity error from an Expansion Bus read access
- an error during a PCI unit access

Refer to the PCI unit specification for the cause of errors during a PCI unit access. The Switching Coprocessor Parity error is valid ONLY for NPE B and is asserted when there is a parity error in Switching coprocessor content addressable memory.



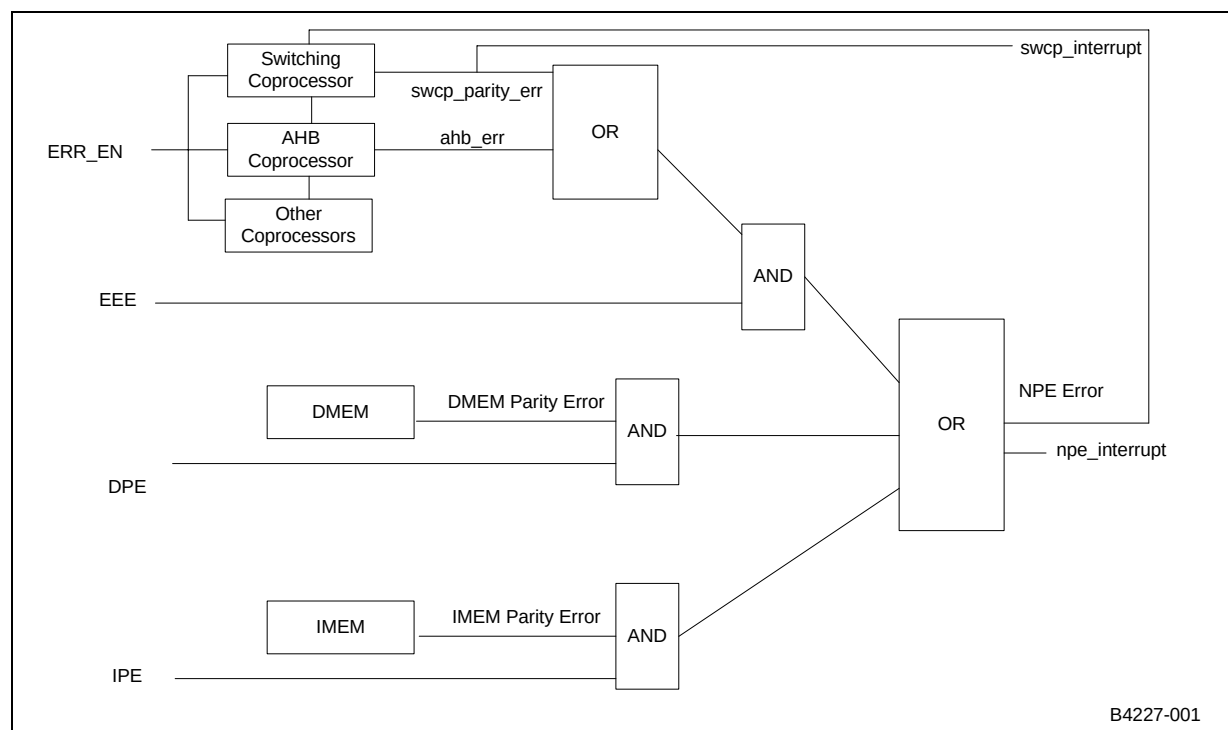
Table 298 on page 953 displays the types of coprocessor errors that can be generated to each NPE contained within the IXP45X/IXP46X network processors.

Table 298. NPE Coprocessor Error

NPE	Defined Coprocessor Errors
NPE A	AHB error
NPE B	AHB error, SWCP Parity error
NPE C	AHB error

Figure 209 on page 953 shows how the different types of errors generate the interrupt to the Intel XScale® Processor and how the errors are communicated back to the coprocessors. Each of these errors; NPE IMEM Parity error, NPE DMEM Parity error and NPE Coprocessor error can be individually masked through control bits as described below. The response of the coprocessors on an occurrence of an error can be controlled by ERR_EN control bit as described below.

Figure 209. NPE Error Handling Illustration



EEE - Control bit to enable stop and interrupt on NPE Coprocessor error. This bit defaults to zero on reset. It is bit 18 of the NPE Core Configuration Bus Control Register.

DPE - Control bit to enable the NPE Core to stop and interrupt on NPE DMEM parity error. It defaults to zero on reset. It is bit 19 of the NPE Core Configuration Bus Control Register.

IPE - Control bit to enable the NPE Core to stop and interrupt on NPE IMEM parity error. It defaults to zero on reset. It is bit 20 of the NPE Core Configuration Bus Control Register.



ERR_EN - Control bit for Coprocessor error response enable. This is Bits 14:12 of configuration register 1 defined in [Chapter 12, "Expansion Bus Controller."](#) It is a software global control bit per NPE and NOT per coprocessor. These control bits are independent of the first three control bits described above in that it dictates the response of each coprocessor as shown in [Table 299 on page 955](#). These bits default to zero on reset.

The following scenario describes a typical Error Handling and recovery mechanism. In all Error Handling scenarios, all three control bits (IPE, DPE, and EEE) must be enabled.

- If there is IMEM, DMEM, or NPE Coprocessor error and the corresponding control bit is enabled, then the NPE core stops execution. The NPE also asserts an interrupt to the Intel XScale processor. In addition, the NPE core will indicate to the Coprocessors that an error happened (NPE error). The error indication to the coprocessors is sticky and will be deasserted only on resetting the NPE core. There are two possible scenarios -
 - SWCP interrupt is asserted in conjunction with NPE interrupt
 - SWCP interrupt is not asserted in conjunction with NPE interrupt

If the SWCP interrupt is asserted in addition to the NPE interrupt, the software must read the Configuration bus status register (in the NPE core). If the External error status bit is set and the IMEM or DMEM parity error status bits are not set, the software can infer a SWCP parity error.

If the SWCP interrupt is not asserted in conjunction with the NPE interrupt, then potentially there are three causes for the interrupt:

- NPE IMEM Parity error
- NPE DMEM Parity error
- AHB error

The software must read the Configuration bus status register to determine if the error which occurred is caused by an NPE IMEM Parity error or a NPE DMEM Parity error. If both the NPE IMEM Parity error and NPE DMEM Parity error bits are not set, then the interrupt must be a result of an AHB error. The causes of AHB error include:

- AHB Slave access error
- Read/Write to an invalid AHB address access
- An illegal transfer size access on the AHB Bus

The AHB Slave access error can be caused by either an uncorrectable ECC error on DDR1 read access, a parity error on a Queue Manager read Access, a parity error from an Expansion Bus read access, or an error during a PCI unit access. Refer to the PCI unit specification for the cause of errors during a PCI unit access. The software can infer an AHB Slave access error if there is an interrupt from the AHB Slave in addition to the NPE interrupt. Else, the software can infer an invalid AHB address access or illegal transfer size access on the AHB.

- Each of the coprocessors behaves uniquely on occurrence of the error. The behavior of the coprocessors is shown in [Table 299 on page 955](#).

**Table 299. NPE Coprocessor Response**

NPE Coprocessor	NPE Error	ERR_EN	NPE Coprocessor Response
AHB Coprocessor			
	Asserted	1	The AHB Coprocessor will lock-up and not accept new transfers.
	Asserted	0	Error happened but the coprocessor continues Normal operation as if no error happened
	Deasserted	X	No error, continues Normal operation
Ethernet Coprocessor			
	Asserted	1	The transmit Enable signal (ecp_tx_en) will be forced to inactive state effectively ending all transmission immediately
	Asserted	0	Error happened but the coprocessor continues Normal operation as if no error happened.
	Deasserted	X	No error, continues Normal operation
UTOPIA Coprocessor			
	Asserted	1	The transmit Enable signal (UTP_OP_FCO) will be forced to inactive state effectively ending all transmission immediately
	Asserted	0	Error happened but the coprocessor continues Normal operation as if no error happened
	Deasserted	X	No error, continues Normal operation
Switching Coprocessor			
	X	X	In case of a parity error, continue operation without locking up. For example, if an error happened during insertion of an entry into the CAM, the SWCP will indicate unsuccessful completion of operation and will not lock-up. But the SWCP will assert an interrupt to the Intel XScale® Processor indicating the parity error.
Other Coprocessors			
	X	X	In case of a parity error, continue operation without locking up.

- Software can reset the errored NPE by writing to Bits 13:11 of the EXP_UNIT_FUSE_REG register defined in the Expansion Bus chapter. Please note resetting the NPE resets the core and all the coprocessors associated with that NPE.
- The Expansion bus unit hardware handshakes with the NPE hardware to indicate that the AHB Bus transactions must be terminated cleanly (and the AHB Bus itself doesn't hang).
- The NPE hardware indicates to the Expansion Bus unit hardware that it is ready to be reset.
- The NPE then goes into reset state.
- The coprocessors on the NPE are also reset. The state of the coprocessors and the state of the pins are described below:



Table 300. NPE Reset State

Coprocessor	State
AES	Reset state
AHB	Reset state
Condition	Reset state
DES	Reset state
Ethernet	The Ethernet Coprocessor will be in reset state. The transmit enable and transmit data pin drives '0' if the coprocessor is configured as MII device. If the coprocessor is configured as a SMII device, the transmit data pin drives '0'. The transmit sync and the transmit clock pin will toggle. The PHY may not recognize the fact that the Ethernet coprocessor is being soft-reset. The assumption is that there is an Upper Layer Protocol recovery in this scenario. The MDIO portion of the Ethernet coprocessor will go into reset state if the NPE B is currently being reset.
FIFO	Reset state
Hash	Reset state
HSS	The HSS Coprocessor will be in reset state. The HSS pins will be tri-stated. So the framer can potentially lose sync with the IXP45X/IXP46X network processors. The recommendation is that the HSS framer is also reset at the same time the NPE is soft reset. This could be accomplished by utilizing the GPIO pins on the IXP45X/IXP46X network processors or through other system mechanisms. The assumption made here is that there is an Upper Layer Protocol recovery in this scenario
NPE Core	The NPE Core is in reset state and the Program Counter is initialized to location 0.
NPE Data Memory (DMEM)	The DMEM must be initialized to restart the NPE.
NPE Instruction Memory (IMEM)	The NPE code must be downloaded to the IMEM to restart the NPE.
Switching Coprocessor	The Switching Coprocessor will be in reset state. The Switching Coprocessor Memory must be re initialized.
UTOPIA Coprocessor	The UTOPIA Coprocessor will be in reset state. The UTOPIA pins will be tri-stated. The recommendation is that the UTOPIA PHY is also reset at the same time the NPE is soft reset. This could be accomplished by utilizing the GPIO pins on the IXP45X/IXP46X network processors or through other system mechanisms. The assumption made here is that there is an Upper Layer Protocol recovery in this scenario. If it is not done, then the recommendation is that we have pull-ups or pull-downs on the board for the transmit enable pin (UTP_OP_FCO) and the Address pins (UTP_OP_ADDR). Multiple PHYs may potentially drive some pins if the pull-ups or pull-downs are not used. Please refer to the <i>Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Datasheet</i> and <i>Intel® IXP45X and Intel® IXP46X Product Line of Network Processors Hardware Design Guidelines</i> for details on the pull-up/pull-downs.

- The reset to the NPE must then be deasserted by writing to bits 13:11 of the EXP_UNIT_FUSE_REG register.
- The instruction firmware for the NPE must be downloaded.
- The data firmware for the NPE must be downloaded.
- The NPE must then be restarted to resume normal operation by writing to the NPE core Execution Control Register.
- The system can then deassert reset to the HSS framer/UTOPIA Device, if applicable.



28.2.3 Expansion Bus Controller Response to Errors

When the Expansion Bus Controller receives an AHB error response, it will return data to the requesting master on the Expansion Bus, but will assert invalid parity on all bytes to let that master know that the data it is receiving is not valid.

28.2.4 Intel XScale® Processor Response to Errors

When the Intel XScale® Processor receives an error on the AHB Bus or on its private MCU interface, it transition to its fault handler. A second such error can irretrievably lose the program counter, causing unpredictable system results. This is the inherent nature of the ARM* architecture which is not altered by the IXP45X/IXP46X network processors.

All other errors reported by the system will be seen by the host processor via interrupt signals to the interrupt controller. As a result of such information, the host processor may attempt to clean up the system on its own, rather than reboot.

If the Intel XScale® Processor receives indication that an NPE has locked up due to errors in internal memories or AHB errors to the AHB coprocessor, the IXP45X/IXP46X network processors may attempt to reset the afflicted NPE, reload its software, and restart. To ensure that the North AHB bus is not rendered unusable, a handshake signal between the AHB coprocessor and the reset mechanism in the Expansion Bus Controller will be implemented to ensure that the AHB coprocessor is idle before allowing reset to propagate to that unit.

An important functional note is that parity errors are possible as a result of reads to the DMEM or IMEM from the APB. In this event, the only indication of an error condition to the source of the request will be an interrupt to the Intel XScale® Processor. Due to this, error handling is only supported for accesses initiated by the Intel XScale® Processor. Other masters, such as the PCI Controller or Expansion Bus Controller, are not supported.

28.2.5 AHB-AHB Bridge Response to Errors

When the AHB-AHB Bridge receives an AHB error on the South AHB, it will respond with an AHB error on the North AHB to the originating master.

28.3 Multiple Error Conditions

It is possible, even likely, that multiple error conditions might occur within the response period of the Intel XScale® Processor. As an example, parity errors in the IMEM or DMEM may result not only in an indication of their condition, but the AHB Coprocessor will error as a result of the NPE beginning its shutdown process. Similarly, an error in the Queue Manager may result not only in an indication of error to the Interrupt Controller, but a resulting error returned to the NPE which is accessing the Queue Manager may result in an error condition in the NPE.

No attempt to validate multiple error scenarios is implemented as a function in hardware. It is the responsibility of the system software to determine an appropriate response to a multiple error scenario.

