

Intel[®] 81341 and 81342 I/O Processors

Developer's Manual

December 2007



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. 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 products are not intended for use in medical, life saving, life sustaining applications.

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® I/O 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>.

AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Create & Share, Intel GigaBlade, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel Play, Intel Play logo, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, RemoteExpress, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside, TokenExpress, VoiceBrick, VTune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The ARM* and ARM Powered logo marks (the ARM marks) are trademarks of ARM, Ltd., and Intel uses these marks under license from ARM, Ltd.

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

Copyright © 2007, Intel Corporation



Contents

1.0	Introduction.....	43
1.1	About This Document.....	43
1.1.1	How To Read This Document	43
1.1.2	Other Relevant Documents	43
1.2	About the Single-Core Intel® 81341 I/O Processor.....	44
1.3	Single-Core Intel® 81341 I/O Processor Features	46
1.4	About the Dual-Core Intel® 81342 I/O Processor.....	47
1.5	Dual-Core Intel® 81342 I/O Processor Features.....	49
1.5.1	Host Interface.....	49
1.5.2	Intel XScale® Processor	49
1.5.3	Internal Busses	50
1.5.4	Application DMA Controller	50
1.5.5	Address Translation Unit	50
1.5.6	Messaging Unit	51
1.5.7	DDR Memory Controller.....	51
1.5.8	SRAM Memory Controller.....	51
1.5.9	Peripheral Bus Interface	51
1.5.10	Performance Monitoring Unit.....	51
1.5.11	I ² C Bus Interface Unit.....	52
1.5.12	UART Unit	52
1.5.13	Interrupt Controller Unit.....	52
1.5.14	Internal Bus System Controller.....	52
1.5.15	Inter-Processor Communication.....	52
1.5.16	Inter-Processor Messaging Unit	52
1.5.17	Timers	52
1.5.18	GPIO	52
1.6	Terminology and Conventions	53
1.6.1	Representing Numbers	53
1.6.2	Fields	53
1.6.3	Specifying Bit and Signal Values.....	54
1.6.4	Signal Name Conventions.....	54
1.6.5	Terminology	54
2.0	Address Translation Unit (PCI-X)	55
2.1	Overview	55
2.2	ATU Address Translation	58
2.2.1	Inbound Transactions	60
2.2.1.1	Inbound Address Translation	61
2.2.1.2	Inbound Write Transaction	64
2.2.1.3	Inbound Read Transaction.....	66
2.2.1.4	Inbound Configuration Cycle Translation	69
2.2.1.5	Discard Timers	71
2.2.2	Outbound Transactions- Single Address Cycle (SAC) Internal Bus Transactions ...	72
2.2.2.1	Outbound Address Translation - Internal Bus Transactions.....	73
2.2.2.2	Outbound Address Translation Windows	74
2.2.2.3	Outbound DMA Transactions.....	76
2.2.3	Outbound Write Transaction	77
2.2.4	Outbound Read Transaction	79
2.2.5	Outbound Configuration Cycle Translation.....	80
2.2.5.1	PCI-X Mode 1 Considerations for Outbound Configuration Cycles	80
2.2.5.2	PCI-X Mode 2 Considerations for Outbound Configuration Cycles	81
2.2.5.3	Outbound Configuration Cycle Error Conditions	81



2.2.6	Internal Bus Operation	82
2.3	Big Endian Byte Swapping.....	83
2.3.1	Inbound Byte Swapping	83
2.3.2	Outbound Byte Swapping	84
2.4	CompactPCI Hot-Swap	85
2.4.1	Pin Interface	85
2.4.1.1	Compact PCI Hot-Swap Mode Select	86
2.5	Expansion ROM Translation Unit.....	87
2.6	ATU Queue Architecture.....	88
2.6.1	Inbound Queues	88
2.6.1.1	Inbound Write Queue Structure	88
2.6.1.2	Inbound Read Queue Structure	89
2.6.1.3	Inbound Delayed Write Queue.....	90
2.6.1.4	Inbound Transaction Queues Command Translation Summary	90
2.6.2	Outbound Queues	91
2.6.2.1	Relaxed Ordering and No Snoop Outbound Request Attributes	91
2.6.3	Transaction Ordering.....	92
2.6.3.1	Transaction Ordering Summary	95
2.6.4	Byte Parity Checking and Generation.....	97
2.6.4.1	Parity Generation	97
2.6.4.2	Parity Checking	98
2.6.4.3	Parity Disabled	98
2.7	ATU Error Conditions	99
2.7.1	Uncorrectable Address and Uncorrectable Attribute Errors on the PCI Interface ..	100
2.7.2	Correctable Address and Correctable Attribute Errors on the PCI Interface....	101
2.7.3	Uncorrectable Data Errors on the PCI Interface	102
2.7.3.1	Outbound Read Request Uncorrectable Data Errors	103
2.7.3.1.1	Immediate Data Transfer	103
2.7.3.1.2	Split Response Termination	104
2.7.3.2	Outbound Write Request Uncorrectable Data Errors	105
2.7.3.2.1	Outbound Writes that are not MSI (Message Signaled Inter-	
	rupts)105	
2.7.3.2.2	MSI Outbound Writes.....	105
2.7.3.3	Inbound Read Completions Uncorrectable Data Errors	106
2.7.3.4	Inbound Configuration Write Completion Message Uncorrectable Data	
	Errors106	
2.7.3.5	Inbound Read Request Uncorrectable Data Errors	106
2.7.3.5.1	Immediate Data Transfer	106
2.7.3.5.2	Split Response Termination	106
2.7.3.6	Inbound Write Request Uncorrectable Data Errors.....	106
2.7.3.7	Outbound Read Completion Uncorrectable Data Errors	107
2.7.3.8	Outbound Split Write Uncorrectable Data Error Message	108
2.7.3.9	Inbound Configuration Write Request.....	109
2.7.3.9.1	Conventional PCI Mode	109
2.7.3.9.2	PCI-X Mode.....	110
2.7.3.10	Split Completion Messages	111
2.7.4	Correctable Data Errors on the PCI Interface	112
2.7.4.1	Inbound Read Request Correctable Data Errors	112
2.7.4.1.1	Immediate Data Transfer	112
2.7.4.1.2	Split Response Termination.....	112
2.7.4.2	Inbound Write Request Correctable Data Errors	112
2.7.4.3	Outbound Read Completion Correctable Data Errors.....	113
2.7.4.4	Inbound Configuration Write Request.....	113
2.7.4.5	Split Completion Messages	113
2.7.5	Master Aborts on the PCI Interface.....	114



2.7.5.1	Master Aborts for Outbound Read or Write Request	114
2.7.5.2	Inbound Read Completion or Inbound Configuration Write Completion Message115	
2.7.5.3	Master-Aborts Signaled by the ATU as a Target.....	115
2.7.5.3.1	Uncorrectable Address Errors	115
2.7.5.3.2	Internal Bus Master-Abort	115
2.7.6	Target Aborts on the PCI Interface.....	116
2.7.6.1	Target Aborts for Outbound Read Request or Outbound Write Request .	116
2.7.6.2	Inbound Read Completion or Inbound Configuration Write Completion Message117	
2.7.6.3	Target-Aborts Signaled by the ATU as a Target	117
2.7.6.3.1	Internal Bus Master Abort.....	117
2.7.6.3.2	Internal Bus Target Abort	117
2.7.6.3.3	Inbound EROM Memory Write	117
2.7.7	Corrupted or Unexpected Split Completions	118
2.7.7.1	Completer Address	118
2.7.7.2	Completer Attributes	118
2.7.8	SERR# Assertion and Detection	119
2.7.9	Internal Bus Error Conditions	120
2.7.9.1	Master Abort on the Internal Bus	120
2.7.9.1.1	Inbound Write Request.....	120
2.7.9.1.2	Inbound Read Request	121
2.7.9.2	Target Abort on the Internal Bus	122
2.7.9.2.1	Conventional Mode	122
2.7.9.2.2	PCI-X Mode	122
2.7.9.3	Parity Error on the Internal Bus	123
2.7.9.3.1	Conventional Mode	123
2.7.9.3.2	PCI-X Mode.....	123
2.7.10	ATU Error Summary	124
2.8	Message-Signaled Interrupts.....	130
2.9	Internal Interrupts	131
2.10	Vital Product Data	132
2.10.1	Configuring Vital Product Data Operation	132
2.10.2	Accessing Vital Product Data	133
2.10.2.1	Reading Vital Product Data	133
2.10.2.2	Writing Vital Product Data	134
2.11	Central Resource Functionality	135
2.11.1	Multi-Function Support.....	135
2.11.2	Inbound Transactions	135
2.11.3	Outbound Transactions	135
2.11.4	PCI Reset (P_RSTOUT#).....	135
2.11.5	PCI Clock Outputs (P_CLKOUT , P_CLKO[3:0])	135
2.11.6	External Clock Driver (CR_FREQ[1:0])	136
2.11.7	Bus Mode and Frequency Initialization.....	137
2.12	Embedded Bridge Functionality	141
2.13	Register Definitions	144
2.13.1	PCI Configuration Registers	144
2.13.2	Internal Bus Registers.....	148
2.13.3	ATU Vendor ID Register - ATUVID	152
2.13.4	ATU Device ID Register - ATUDID	152
2.13.5	ATU Command Register - ATUCMD	153
2.13.6	ATU Status Register - ATUSR.....	154
2.13.7	ATU Revision ID Register - ATURID.....	156
2.13.8	ATU Class Code Register - ATUCCR.....	156
2.13.9	ATU Cacheline Size Register - ATUCLSR.....	157



2.13.10	ATU Latency Timer Register - ATULT	157
2.13.11	ATU Header Type Register - ATUHTR	158
2.13.12	ATU BIST Register - ATUBISTR	159
2.13.13	Inbound ATU Base Address Register 0 - IABAR0	160
2.13.14	Inbound ATU Upper Base Address Register 0 - IAUBAR0	161
2.13.15	Inbound ATU Base Address Register 1 - IABAR1	162
2.13.16	Inbound ATU Upper Base Address Register 1 - IAUBAR1	163
2.13.17	Inbound ATU Base Address Register 2 - IABAR2	164
2.13.18	Inbound ATU Upper Base Address Register 2 - IAUBAR2	165
2.13.19	ATU Subsystem Vendor ID Register - ASVIR	166
2.13.20	ATU Subsystem ID Register - ASIR	166
2.13.21	Expansion ROM Base Address Register - ERBAR	167
2.13.22	ATU Capabilities Pointer Register - ATU_Cap_Ptr	168
2.13.23	Determining Block Sizes for Base Address Registers	169
2.13.24	ATU Interrupt Line Register - ATUILR	171
2.13.25	ATU Interrupt Pin Register - ATUIPR	172
2.13.26	ATU Minimum Grant Register - ATUMGNT	172
2.13.27	ATU Maximum Latency Register - ATUMLAT	173
2.13.28	Inbound ATU Limit Register 0 - IALR0	174
2.13.29	Inbound ATU Translate Value Register 0 - IATVR0	175
2.13.30	Inbound ATU Upper Translate Value Register 0 - IAUTVR0	175
2.13.31	Inbound ATU Limit Register 1 - IALR1	176
2.13.32	Inbound ATU Translate Value Register 1 - IATVR1	177
2.13.33	Inbound ATU Upper Translate Value Register 1 - IAUTVR1	177
2.13.34	Inbound ATU Limit Register 2 - IALR2	178
2.13.35	Inbound ATU Translate Value Register 2 - IATVR2	179
2.13.36	Inbound ATU Upper Translate Value Register 2 - IAUTVR2	179
2.13.37	Expansion ROM Limit Register - ERLR	180
2.13.38	Expansion ROM Translate Value Register - ERTVR	181
2.13.39	Expansion ROM Upper Translate Value Register - ERUTVR	181
2.13.40	ATU Configuration Register - ATUCR	182
2.13.41	PCI Configuration and Status Register - PCSR	183
2.13.42	ATU Interrupt Status Register - ATUISR	186
2.13.43	ATU Interrupt Mask Register - ATUIMR	188
2.13.44	VPD Capability Identifier Register - VPD_Cap_ID	190
2.13.45	VPD Next Item Pointer Register - VPD_Next_Item_Ptr	190
2.13.46	VPD Address Register - VPDAR	191
2.13.47	VPD Data Register - VPDDR	191
2.13.48	PM Capability Identifier Register - PM_Cap_ID	192
2.13.49	PM Next Item Pointer Register - PM_Next_Item_Ptr	192
2.13.50	ATU Power Management Capabilities Register - APMCR	193
2.13.51	ATU Power Management Control/Status Register - APMCSR	194
2.13.52	ATU Scratch Pad Register - ATUSPR	195
2.13.53	PCI-X Capability Identifier Register - PCI-X_Cap_ID	195
2.13.54	PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr	196
2.13.55	PCI-X Command Register - PCIXCMD	196
2.13.56	PCI-X Status Register - PCIXSR	198
2.13.57	ECC Control and Status Register - ECCCSR	200
2.13.58	ECC First Address Register - ECCFAR	203
2.13.59	ECC Second Address Register - ECCSAR	204
2.13.60	ECC Attribute Register - ECCAR	205
2.13.61	CompactPCI Hot-Swap Capability ID Register	205
2.13.62	Offset EDh: HS_NXTP - Next Item Pointer	206
2.13.63	HS_CNTRL - Hot-Swap Control/Status Register	207
2.13.64	Inbound ATU Base Address Register 3 - IABAR3	209



2.13.65	Inbound ATU Upper Base Address Register 3 - IAUBAR3	210
2.13.66	Inbound ATU Limit Register 3 - IALR3	211
2.13.67	Inbound ATU Translate Value Register 3 - IATVR3	212
2.13.68	Inbound ATU Upper Translate Value Register 3 - IAUTVR3	212
2.13.69	Outbound I/O Base Address Register - OIOBAR	213
2.13.70	Outbound I/O Window Translate Value Register - OIOWTVR	214
2.13.71	Outbound Upper Memory Window Base Address Register 0 - OUMBAR0	215
2.13.72	Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0	216
2.13.73	Outbound Upper Memory Window Base Address Register 1 - OUMBAR1	217
2.13.74	Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1	218
2.13.75	Outbound Upper Memory Window Base Address Register 2 - OUMBAR2	219
2.13.76	Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2	220
2.13.77	Outbound Upper Memory Window Base Address Register 3 - OUMBAR3	221
2.13.78	Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3	222
2.13.79	Outbound Configuration Cycle Address Register - OCCAR	223
2.13.80	Outbound Configuration Cycle Data Register - OCCDR	224
2.13.81	Outbound Configuration Cycle Function Number - OCCFN	224
2.13.82	PCI Interface Error Control and Status Register - PIECSR	225
2.13.83	PCI Interface Error Address Register - PCIEAR	226
2.13.84	PCI Interface Error Upper Address Register - PCIEUAR	227
2.13.85	PCI Interface Error Context Address Register — PCIECAR	228
2.13.86	Internal Arbiter Control Register - IACR	229
2.13.87	Multi-Transaction Timer - MTT	230
2.13.88	PCIX RCOMP Control Register — PCR	231
2.13.89	PCIX Pad ODT Drive Strength Manual Override Values Registers — PPDSMOVR	232
2.13.90	PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V/1.5 V Switch Supply Voltage) — PPDSMOVR3.3_1.5233	
2.13.91	PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V Dedicated Supply Voltage) — PPDSMOVR3.3234	
3.0	Address Translation Unit (PCI Express)	235
3.1	Overview	235
3.2	PCI Express Link Characteristics	238
3.3	ATU Address Translation	239
3.3.1	Inbound Transactions	242
3.3.1.1	Inbound Address Translation	242
3.3.1.2	Inbound Memory Write Transaction	245
3.3.1.3	Inbound Memory Read Transaction	246
3.3.1.4	Inbound I/O Cycle Translation	247
3.3.1.5	Inbound Configuration Cycle Translation (ID Routed)	247
3.3.1.6	Inbound Vendor_Defined Message Transactions	248
3.3.2	Outbound Transactions	249
3.3.2.1	Outbound Address Translation - Internal Bus Transactions	250
3.3.2.2	Outbound Address Translation Windows	251
3.3.2.3	Outbound DMA Transactions	254
3.3.2.4	Outbound Function Number	254
3.3.3	Outbound Write Transaction	255
3.3.4	Outbound Read Transaction	256
3.3.5	Outbound Configuration Cycle Translation	257
3.3.5.1	Outbound Configuration Cycle Error Conditions	257
3.3.5.2	Outbound Configuration Completions with Retry Status (CRS)	257
3.3.5.3	Outbound PCI Express Message Transactions	258



3.3.5.4	Completion Timeout Mechanism	258
3.4	Big Endian Byte Swapping.....	259
3.4.1	Inbound Byte Swapping	259
3.4.2	Outbound Byte Swapping.....	260
3.5	Messaging Unit.....	261
3.6	PCI Express Messages	262
3.7	Expansion ROM Translation Unit.....	264
3.8	ATU Queue Architecture.....	265
3.8.1	Inbound Queues	265
3.8.1.1	Inbound Posted Queue Structure	265
3.8.1.2	Inbound Non Posted Queue Structure	266
3.8.1.3	Inbound Completion Queue Structure	266
3.8.1.4	Inbound Transaction Queues Command Translation Summary	266
3.8.2	Outbound Queues.....	267
3.8.2.1	Relaxed Ordering and No Snoop Outbound Request Attributes	267
3.8.3	Transaction Ordering.....	268
3.8.3.1	Transaction Ordering Summary	271
3.8.4	Byte Parity Checking and Generation.....	272
3.8.4.1	Parity Generation	272
3.8.4.2	Parity Checking	273
3.8.4.3	Parity Disabled.....	273
3.9	ATU Error Conditions.....	274
3.9.1	PCI Express Errors	275
3.9.1.1	Role Based Error Reporting.....	275
3.9.1.2	Malformed Packets	276
3.9.1.3	ECRC Check Failed	276
3.9.1.4	Unsupported Request.....	277
3.9.1.5	Completer Abort.....	277
3.9.1.6	Unexpected Completions	277
3.9.1.7	Poisoned TLP Received	278
3.9.1.8	Completion Timeout	278
3.9.2	Parity Error on the Internal Bus	279
3.9.3	ATU Error Summary	279
3.10	PCI Express Hot-Plug Support	283
3.11	Reset.....	284
3.12	Message-Signaled Interrupts	285
3.12.1	Legacy Interrupts	285
3.12.2	Internal Interrupts	285
3.13	Vital Product Data.....	286
3.13.1	Configuring Vital Product Data Operation	286
3.13.2	Accessing Vital Product Data	287
3.13.2.1	Reading Vital Product Data	287
3.13.2.2	Writing Vital Product Data	288
3.14	Root Complex Functionality	289
3.15	Embedded Bridge Functionality.....	290
3.16	Register Definitions.....	293
3.16.1	Extended Capabilities Registers	294
3.16.2	Internal Bus Addresses.....	297
3.16.3	ATU Vendor ID Register - ATUVID	301
3.16.4	ATU Device ID Register - ATUDID	301
3.16.5	ATU Command Register - ATUCMD.....	302
3.16.6	ATU Status Register - ATUSR	303
3.16.7	ATU Revision ID Register - ATURID	304
3.16.8	ATU Class Code Register - ATUCCR	304
3.16.9	ATU Cacheline Size Register - ATUCLSR.....	305
3.16.10	ATU Latency Timer Register - ATULT	305



3.16.11	ATU Header Type Register - ATUHTR.....	306
3.16.12	ATU BIST Register - ATUBISTR.....	307
3.16.13	Inbound ATU Base Address Register 0 - IABAR0	308
3.16.14	Inbound ATU Upper Base Address Register 0 - IAUBAR0	309
3.16.15	Determining Block Sizes for Base Address Registers	310
3.16.16	Inbound ATU Base Address Register 1 - IABAR1	312
3.16.17	Inbound ATU Upper Base Address Register 1 - IAUBAR1	313
3.16.18	Inbound ATU Base Address Register 2 - IABAR2	314
3.16.19	Inbound ATU Upper Base Address Register 2 - IAUBAR2	315
3.16.20	ATU Subsystem Vendor ID Register - ASVIR	316
3.16.21	ATU Subsystem ID Register - ASIR	316
3.16.22	Expansion ROM Base Address Register - ERBAR.....	317
3.16.23	ATU Capabilities Pointer Register - ATU_Cap_Ptr.....	318
3.16.24	ATU Interrupt Line Register - ATUILR	319
3.16.25	ATU Interrupt Pin Register - ATUIPR	320
3.16.26	ATU Minimum Grant Register - ATUMGNT	320
3.16.27	ATU Maximum Latency Register - ATUMLAT	321
3.16.28	Inbound ATU Limit Register 0 - IALR0	322
3.16.29	Inbound ATU Translate Value Register 0 - IATVR0	323
3.16.30	Inbound ATU Upper Translate Value Register 0 - IAUTVR0	323
3.16.31	Inbound ATU Limit Register 1 - IALR1	324
3.16.32	Inbound ATU Translate Value Register 1 - IATVR1	325
3.16.33	Inbound ATU Upper Translate Value Register 1 - IAUTVR1	325
3.16.34	Inbound ATU Limit Register 2 - IALR2	326
3.16.35	Inbound ATU Translate Value Register 2 - IATVR2	327
3.16.36	Inbound ATU Upper Translate Value Register 2 - IAUTVR2	328
3.16.37	Expansion ROM Limit Register - ERLR.....	328
3.16.38	Expansion ROM Translate Value Register - ERTVR.....	329
3.16.39	Expansion ROM Upper Translate Value Register - ERUTVR.....	329
3.16.40	ATU Configuration Register - ATUCR	330
3.16.41	PCI Configuration and Status Register - PCSR	331
3.16.42	ATU Interrupt Status Register - ATUISR	333
3.16.43	ATU Interrupt Mask Register - ATUIMR.....	336
3.16.44	PCI Express Message Control/Status Register - PEMCSR	337
3.16.45	PCI Express Link Control/Status Register - PELCSR	338
3.16.46	VPD Capability Identifier Register - VPD_Cap_ID	339
3.16.47	VPD Next Item Pointer Register - VPD_Next_Item_Ptr.....	339
3.16.48	VPD Address Register - VPDAR.....	340
3.16.49	VPD Data Register - VPDDR.....	340
3.16.50	PM Capability Identifier Register - PM_Cap_ID.....	341
3.16.51	PM Next Item Pointer Register - PM_Next_Item_Ptr	341
3.16.52	ATU Power Management Capabilities Register - APMCR.....	342
3.16.53	ATU Power Management Control/Status Register - APMCSR	343
3.16.54	ATU Scratch Pad Register - ATUSPR.....	344
3.16.55	PCI Express Capability List Register - PCIE_CAPID	344
3.16.56	PCI Express Next Item Pointer Register - PCIE_NXTP	345
3.16.57	PCI Express Capabilities Register - PCIE_CAP.....	346
3.16.58	PCI Express Device Capabilities Register - PCIE_DCAP.....	347
3.16.59	PCI Express Device Control Register - PE_DCTL.....	348
3.16.60	PCI Express Device Status Register - PE_DSTS.....	350
3.16.61	PCI Express Link Capabilities Register - PE_LCAP.....	351
3.16.62	PCI Express Link Control Register - PE_LCTL	352
3.16.63	PCI Express Link Status Register - PE_LSTS.....	353
3.16.64	PCI Express Slot Capabilities Register - PE_SCAP.....	354
3.16.65	PCI Express Slot Control Register - PE_SCR	355



3.16.66	PCI Express Slot Status Register - PE_SSTS	356
3.16.67	PCI Express Root Control Register - PE_RCR.....	357
3.16.68	PCI Express Root Status Register - PE_RSR.....	358
3.16.69	PCI Express Advanced Error Capability Identifier - ADVERR_CAPID.....	359
3.16.70	PCI Express Uncorrectable Error Status - ERRUNC_STS.....	360
3.16.71	PCI Express Uncorrectable Error Mask - ERRUNC_MSK.....	361
3.16.72	PCI Express Uncorrectable Error Severity - ERRUNC_SEV	362
3.16.73	PCI Express Correctable Error Status - ERRCOR_STS.....	363
3.16.74	PCI Express Correctable Error Mask - ERRCOR_MSK.....	364
3.16.75	Advanced Error Control and Capability Register - ADVERR_CTL.....	365
3.16.76	PCI Express Advanced Error Header Log - ADVERR_LOG0	365
3.16.77	PCI Express Advanced Error Header Log - ADVERR_LOG1	366
3.16.78	PCI Express Advanced Error Header Log - ADVERR_LOG2	366
3.16.79	PCI Express Advanced Error Header Log - ADVERR_LOG3	367
3.16.80	Root Error Command Register - RERR_CMD	367
3.16.81	Root Error Status Register	368
3.16.82	Error Source Identification Register - RERR_ID	369
3.16.83	Device Serial Number Capability - DSN_CAP.....	369
3.16.84	Device Serial Number Lower DW Register - DSN_LDW	370
3.16.85	Device Serial Number Upper DW Register - DSN_UDW.....	370
3.16.86	PCI Express Advisory Error Control Register - PIE_AEC	371
3.16.87	Power Budgeting Enhanced Capability Header - PWRBGT_CAPID.....	372
3.16.88	Power Budgeting Data Select Register - PWRBGT_DSEL	372
3.16.89	Power Budgeting Data Register - PWRBGT_DATA	373
3.16.90	Power Budgeting Capability Register - PWRBGT_CAP	374
3.16.91	Power Budgeting Information Registers[0:23]—PWRBGT_INFO[0:23]	375
3.16.92	Outbound I/O Base Address Register - OIOBAR.....	376
3.16.93	Outbound I/O Window Translate Value Register - OIOWTVR	377
3.16.94	Outbound Upper Memory Window Base Address Register 0 - OUMBAR0.....	378
3.16.95	Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0	379
3.16.96	Outbound Upper Memory Window Base Address Register 1 - OUMBAR1.....	380
3.16.97	Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1	381
3.16.98	Outbound Upper Memory Window Base Address Register 2 - OUMBAR2.....	382
3.16.99	Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2	383
3.16.100	Outbound Upper Memory Window Base Address Register 3 - OUMBAR3	384
3.16.101	Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3.....	385
3.16.102	Outbound Configuration Cycle Address Register - OCCAR.....	386
3.16.103	Outbound Configuration Cycle Data Register - OCCDR.....	387
3.16.104	Outbound Configuration Cycle Function Number - OCCFN	388
3.16.105	Inbound Vendor Message Header Register 0 - IVMHR0	389
3.16.106	Inbound Vendor Message Header Register 1 - IVMHR1	390
3.16.107	Inbound Vendor Message Header Register 2 - IVMHR2	391
3.16.108	Inbound Vendor Message Header Register 3 - IVMHR3	392
3.16.109	Inbound Vendor Message Payload Register - IVMPR	392
3.16.110	Outbound Vendor Message Header Register 0 - OVMHR0	393
3.16.111	Outbound Vendor Message Header Register 1 - OVMHR1	394
3.16.112	Outbound Vendor Message Header Register 2 - OVMHR2	395
3.16.113	Outbound Vendor Message Header Register 3 - OVMHR3	395
3.16.114	Outbound Vendor Message Payload Register - OVMPR.....	396
3.16.115	PCI Interface Error Control and Status Register - PIE_CSR.....	397
3.16.116	PCI Interface Error Status - PIE_STS	398



3.16.117	PCI Interface Error Mask - PIE_MSK	399
3.16.118	PCI Interface Error Header Log - PIE_LOG0	400
3.16.119	PCI Interface Error Header Log 1 - PIE_LOG1	400
3.16.120	PCI Interface Error Header Log 2 - PIE_LOG2	401
3.16.121	PCI Interface Error Header Log - PIE_LOG3	401
3.16.122	PCI Interface Error Descriptor Log	402
3.16.123	ATU Reset Control Register - ATURCR	402
4.0	Messaging Unit	403
4.1	Overview	403
4.2	Theory of Operation	404
4.2.1	Transaction Ordering	407
4.3	Message Registers	408
4.3.1	Outbound Messages	408
4.3.2	Inbound Messages	408
4.4	Doorbell Registers	409
4.4.1	Outbound Doorbells	409
4.4.2	Inbound Doorbells	409
4.5	Circular Queues	410
4.5.1	Inbound Free Queue	414
4.5.2	Inbound Post Queue	415
4.5.3	Outbound Post Queue	416
4.5.4	Outbound Free Queue	417
4.6	Index Registers	419
4.7	Messaging Unit Error Conditions	420
4.8	Message-Signaled Interrupts	421
4.8.1	MSI Capability Structure	421
4.8.2	MSI-X Capability and Table Structures	422
4.8.3	Level-Triggered Versus Edge-Triggered Interrupts	424
4.9	Register Definitions	425
4.9.1	Inbound Message Register - IMRx	427
4.9.2	Outbound Message Register - OMRx	427
4.9.3	Inbound Doorbell Register - IDR	428
4.9.4	Inbound Interrupt Status Register - IISR	429
4.9.5	Inbound Interrupt Mask Register - IIMR	430
4.9.6	Outbound Doorbell Register - ODR	431
4.9.7	Outbound Interrupt Status Register - OISR	432
4.9.8	Outbound Interrupt Mask Register - OIMR	433
4.9.9	Inbound Reset Control and Status Register - IRCSR	434
4.9.10	Outbound Reset Control and Status Register - ORCSR	435
4.9.11	MSI Inbound Message Register — MIMR	436
4.9.12	MU Configuration Register - MUCR	437
4.9.13	Queue Base Address Register - QBAR	438
4.9.14	Inbound Free Head Pointer Register - IFHPR	439
4.9.15	Inbound Free Tail Pointer Register - IFTPR	439
4.9.16	Inbound Post Head Pointer Register - IPHPR	440
4.9.17	Inbound Post Tail Pointer Register - IPTPR	440
4.9.18	Outbound Free Head Pointer Register - OFHPR	441
4.9.19	Outbound Free Tail Pointer Register - OFTPR	441
4.9.20	Outbound Post Head Pointer Register - OPHPR	442
4.9.21	Outbound Post Tail Pointer Register - OPTPR	442
4.9.22	Index Address Register - IAR	443
4.9.23	MU Base Address Register - MUBAR	444
4.9.24	MU Upper Base Address Register - MUUBAR	445
4.9.25	MU MSI-X Table Message Address Registers - M_MT_MAR[0:7]	446



4.9.26	MU MSI-X Table Message Upper Address Registers - M_MT_MUAR[0:7]	447
4.9.27	MU MSI-X Table Message Data Registers - M_MT_MDR[0:7]	448
4.9.28	MU MSI-X Table Message Vector Control Registers - M_MT_MVCR[0:7]	449
4.9.29	MU MSI-X Pending Bits Array Register - M_MPBAR	450
4.9.30	MSI Capability Identifier Register - Cap_ID	450
4.9.31	MSI Next Item Pointer Register - MSI_Next_Ptr	451
4.9.32	Message Control Register - Message_Control	452
4.9.33	Message Address Register - Message_Address	453
4.9.34	Message Upper Address Register - Message_Upper_Address	454
4.9.35	Message Data Register- Message_Data	455
4.9.36	MSI-X Capability Identifier Register - MSI-X_Cap_ID	456
4.9.37	MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr	457
4.9.38	MSI-X Message Control Register - MSI-X_MCR	458
4.9.39	MSI-X Table Offset Register — MSI-X_Table_Offset	459
4.9.40	MSI-X Pending Bit Array Offset Register - MSI-X_PBA_Offset	460
4.9.41	MU MSI-X Control Register X — MMCRx	461
4.9.42	Inbound MSI Interrupt Pending Register x — IMIPRx	462
4.10	Power/Default Status	462
5.0	Application DMA Unit	463
5.1	Overview	463
5.2	Theory of Operation	464
5.3	ADMA Descriptors	466
5.3.1	Basic Chain Descriptor Format	467
5.3.2	Full Chain Descriptor Format	469
5.3.3	P+Q Chain Descriptor Format	472
5.3.4	Dual XOR Descriptor Format	476
5.3.5	P+Q Update Descriptor Format	478
5.4	ADMA Descriptor Processing	480
5.4.1	Scatter Gather Transfers	482
5.4.2	Synchronizing a Program to Chained Operation	482
5.4.3	Appending to the End of a Chain	484
5.5	Data Queues	485
5.5.1	Data Transfer Options	485
5.6	Data Alignment	487
5.6.1	64-bit Unaligned Data Transfers	487
5.6.2	64/32-bit Unaligned Data Transfers	488
5.7	ADMA Operations	489
5.7.1	Per Channel Support for ADMA Operations	490
5.7.2	XOR Operation	491
5.7.3	XOR Operation with P+Q RAID-6	494
5.7.4	Dual XOR Operation	497
5.7.5	P+Q Update Operation	500
5.7.6	Zero Result Buffer Check	502
5.7.7	P+Q Zero Result Buffer Check	503
5.7.8	Memory Block Fill Operation	505
5.8	CRC Generation	506
5.8.1	CRC Mode Configuration and Operation	506
5.8.1.1	Chaining of CRC Calculation Across Multiple Descriptors	507
5.8.2	CRC-32C Algorithm	508
5.9	Programming Model State Diagram	509
5.10	Application DMA Priority	510
5.11	Programming the Application DMA	511
5.11.1	Application DMA Initialization	511
5.11.2	Start Transfer	511



5.11.3	Suspend Application DMA	512
5.12	Internal Interface Parity Checking and Generation	513
5.12.1	Address and Data Parity Generation	514
5.12.1.1	Address Parity Generation	514
5.12.1.2	Data Parity Generation	515
5.12.2	Address and Data Parity Checking	516
5.12.2.1	Address Parity Checking	516
5.12.2.2	Data Parity Checking	517
5.12.2.3	Parity Disabled	517
5.12.2.4	Parity Testing	517
5.13	Interrupts	518
5.14	Error Conditions	519
5.14.1	Fatal Errors	519
5.14.2	Non-Fatal Errors	519
5.15	Power-Up/Default Status	519
5.16	Register Definitions	520
5.16.1	ADMA Channel Control Register x — ACCRx	521
5.16.2	ADMA Channel Status Register x — ACSRx	522
5.16.3	ADMA Descriptor Address Register x — ADARx	524
5.16.4	Internal Interface Parity Control Register x — IIPCRx	524
5.16.5	ADMA Next Descriptor Address Register x — ANDARx	525
5.16.6	ADMA Descriptor Control Register x — ADCRx	526
5.16.7	CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx	529
5.16.8	ADMA Byte Count Register x — ABCRx	530
5.16.9	Destination Lower Address / P_Destination Lower Address Register x — DLADRx 531	
5.16.10	Destination Upper Address / PQ_Destination Upper Address Register x — DUADRx	532
5.16.11	Source Lower Address Register 0...15_x — SLAR0...15_x	533
5.16.12	Source Upper Address Register 0...15_x — SUAR0...15_x	535
6.0	System Controller (SC) and Internal Bus Bridge	537
6.1	Overview	537
6.2	Theory of Operation	538
6.2.1	System Controller	538
6.2.2	Internal Bus Requester IDs	538
6.2.3	Parity Testing	539
6.3	Internal Bus Bridge	541
6.3.1	Theory of Operation	542
6.3.2	Internal Bus Commands	543
6.3.3	Transaction Queues	543
6.3.4	Bridge Memory Window	544
6.3.5	Ordering and Passing Rules	545
6.3.5.1	Strong Ordering Rule Requirements	545
6.3.6	Parity Support	546
6.3.6.1	Address Parity Generation	546
6.3.6.2	Address Parity Checking	546
6.3.6.3	Data Parity on Outbound Transactions	546
6.3.6.4	Data Parity on Inbound Transactions	546
6.3.7	Error Detection and Handling	547
6.3.7.1	Bridge North Internal Bus Interface Error	547
6.3.7.2	Bridge South Internal Bus Interface Error	548
6.4	System Controller Register Definitions	549
6.5	Internal Bus Bridge Register Definitions	550
6.5.1	Internal Bus Arbitration Control Register — IBACR	551



6.5.2	South Internal Bus Address Test Control Register — SIBATCR.....	553
6.5.3	South Internal Bus Data Test Control Register — SIBDTCR.....	554
6.5.4	Peripheral Memory-Mapped Register Base Address Register — PMMRBAR.....	555
6.5.5	Determining Block Sizes for Memory Windows	556
6.5.6	Bridge Window Base Address Register — BWBAR	557
6.5.7	Bridge Window Upper Base Address Register — BWUBAR	558
6.5.8	Bridge Window Limit Register — BWLR	559
6.5.9	Bridge Error Control and Status Register — BECSR.....	560
6.5.10	Bridge Error Address Register — BERAR.....	562
6.5.11	Bridge Error Upper Address Register — BERUAR.....	562
7.0	DDR SDRAM Memory Controller	563
7.1	Overview	563
7.2	Glossary.....	564
7.3	Theory of Operation	565
7.3.1	Functional Blocks	565
7.3.1.1	Transaction Ports	566
7.3.1.1.1	North Internal Bus Port.....	566
7.3.1.1.2	South Internal Bus Port.....	566
7.3.1.1.3	Application DMA Ports	566
7.3.1.1.4	Messaging Unit Port.....	566
7.3.1.2	Address Decode Blocks	567
7.3.1.2.1	DDR SDRAM Memory Array Space.....	567
7.3.1.2.2	Memory-Mapped Register Space	567
7.3.1.2.3	North Internal Bus Port Address Decode	567
7.3.1.2.4	South Internal Bus Port Address Decode	567
7.3.1.2.5	Application DMA Port Address Decode	567
7.3.1.3	Memory Transaction Queues	568
7.3.1.3.1	North Internal Bus Port Transaction Queue (NIBPTQ)	568
7.3.1.3.2	South Internal Bus Port Transaction Queue (SIBPTQ).....	568
7.3.1.4	Configuration Registers	568
7.3.1.5	Refresh Counter	568
7.3.1.6	Memory Controller Arbiter (DMARB).....	568
7.3.1.7	DDR SDRAM Control Block.....	569
7.3.1.7.1	Page Control Block	569
7.3.1.7.2	DDR SDRAM State Machine and Pipeline Queues	569
7.3.1.7.3	Error Correction Logic	569
7.3.2	DMCU Arbitration and Configuration	570
7.3.2.1	DMCU Port Priority.....	570
7.3.2.2	DMCU Port Transaction Count	570
7.3.2.3	North Internal Bus Port Preemption	571
7.3.2.4	North Internal Bus Port Transaction Ordering	571
7.3.2.5	South Internal Bus Port Ordering	571
7.3.2.6	Application DMAs Port Ordering	571
7.3.2.7	DMCU Port Coherency	571
7.3.3	DDR SDRAM Memory Support.....	572
7.3.3.1	DDR SDRAM Interface	572
7.3.3.2	DDR SDRAM Addressing	575
7.3.3.3	DDR SDRAM Bank Sizes and Configurations	578
7.3.3.4	32-bit Data Bus Width.....	588
7.3.3.5	Page Hit/Miss Determination	589
7.3.3.6	DDR SDRAM Commands.....	593
7.3.3.7	DDR SDRAM Initialization	594
7.3.3.8	DDR SDRAM Mode Programming	601
7.3.3.9	DDR SDRAM Read Cycle	603
7.3.3.10	DDR SDRAM Write Cycle	604
7.3.3.11	DDR SDRAM Refresh Cycle	606



7.3.4	DDR Error Correction and Detection	607
7.3.4.1	DDR ECC Generation	608
7.3.4.2	DDR ECC Generation for Partial Writes	609
7.3.4.3	DDR ECC Checking	611
7.3.4.4	Scrubbing	615
7.3.4.4.1	DDR ECC Example Using H-Matrix	615
7.3.4.5	DDR ECC Disabled	616
7.3.4.6	DDR ECC Testing	616
7.3.5	Memory Regions	616
7.3.6	DDR SDRAM Clocking	616
7.4	Power Failure Mode	617
7.4.1	Theory of Operation	617
7.4.2	Power Failure Sequence	618
7.4.2.1	Power Failure Impact on the System	618
7.4.2.2	System Assumptions	618
7.4.3	Memory Controller Response to P_RST#	619
7.5	ECC Interrupts/Error Conditions	621
7.5.1	Single-Bit Error Detection	622
7.5.2	Multi-bit Error Detection	623
7.6	Parity Interrupts/Error Conditions	624
7.7	Reset Conditions	624
7.8	Register Definitions	625
7.8.1	SDRAM Initialization Register — SDIR	627
7.8.2	SDRAM Control Register 0 — SDCR0	628
7.8.3	SDRAM Control Register 1 — SDCR1	630
7.8.4	SDRAM Base Register — SDBR	632
7.8.5	SDRAM Upper Base Register — SDUBR	633
7.8.6	Secondary SDRAM Base Register - SSDBR	634
7.8.7	SDRAM Bank Size Register — SBSR	635
7.8.8	SDRAM 32-bit Region Size Register — S32SR	637
7.8.9	DDR ECC Control Register — DECCR	638
7.8.10	DDR ECC Log Registers — DELOG0, DELOG1	639
7.8.11	DDR ECC Address Registers — DEAR0, DEAR1	641
7.8.12	DDR ECC Context Address Registers — DECAR0, DECAR1	642
7.8.13	DDR ECC Context Upper Address Registers — DECUAR0, DECUAR1	643
7.8.14	DDR ECC Test Register — DECTST	643
7.8.15	DDR Parity Control and Status Register — DPCSR	644
7.8.16	DDR Parity Address Register — DPAR	645
7.8.17	DDR Parity Upper Address Register — DPUAR	645
7.8.18	DDR Parity Context Address Register — DPCAR	646
7.8.19	DDR Parity Context Upper Address Register — DPCUAR	646
7.8.20	DDR Memory Controller Interrupt Status Register — DMCISR	647
7.8.21	DMCU Port Transaction Count Register — DMPTCR	648
7.8.22	DMCU Preemption Control Register — DMPCR	650
7.8.23	Frequency Register — RFR	651
7.8.24	DDR RCOMP Control Register — DRCR	652
7.8.25	RCOMP Pad Drive Strength Select Register — RPDSR	654
7.8.26	DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR ..	655
7.8.27	DQ Pad Drive Strength Manual Override Values Register — DQPDSR	656
7.8.28	MA Pad Drive Strength Manual Override Values Register — MAPDSR	657
7.8.29	MCLK Pad Drive Strength Manual Override Values Register — MPDSR	658
7.8.30	CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR....	659
7.8.31	DLL Delay Register 0 — DLLR0	660
7.8.32	DLL Delay Register 1 — DLLR1	661
7.8.33	DLL Delay Register 2 — DLLR2	662



7.8.34	DLL Delay Register 3 — DLLR3	663
7.8.35	DLL Delay Register 4 — DLLR4	664
7.8.36	DLL Delay for Receive Enable Register — DLLRCVER.....	665
7.8.36.1	Determining DLLRCVEREN Pulse Optimum Location.....	668
7.8.36.1.1	DLLRCVREN Algorithm	669
7.8.37	DDR RCOMP Status Register for 15 Ohm RCOMP — DRSR15.....	672
7.8.38	DDR RCOMP Status Register for 17.9 Ohm RCOMP — DRSR17	672
7.8.39	DDR RCOMP Status Register for 25 Ohm RCOMP — DRSR25	673
7.8.40	DDR RCOMP Status Register for 35 Ohm RCOMP — DRSR35.....	673
7.8.41	DDR RCOMP Status Register for 50 Ohm RCOMP — DRSR50.....	674
8.0	SRAM Memory Controller	675
8.1	Overview	675
8.2	Glossary.....	676
8.3	Theory of Operation	677
8.3.1	Functional Block	677
8.3.1.1	Transaction Ports	677
8.3.1.1.1	North Internal Bus Ports.....	677
8.3.1.2	Address Decode Blocks	678
8.3.1.2.1	SRAM Memory Array Space	678
8.3.1.2.2	Memory-Mapped Register Space	678
8.3.1.2.3	North Internal Bus Port Address Decode.....	678
8.3.1.3	Memory Transaction Queues	678
8.3.1.3.1	North Internal Bus Port Transaction Queue (NIBPTQ)	678
8.3.1.4	Configuration Registers	678
8.3.1.5	SRAM Control Block	678
8.3.1.5.1	SRAM State Machine and Pipeline Queues	678
8.3.1.5.2	Error Correction Logic.....	679
8.3.1.6	North Internal Bus Port Transaction Ordering	680
8.3.1.7	SMCU Port Coherency	680
8.3.2	SRAM Memory Interface Support	681
8.3.2.1	SRAM Initialization	681
8.3.2.2	SRAM Read Sequence	681
8.3.2.3	SRAM Write Sequence.....	682
8.3.3	Error Correction and Detection	683
8.3.3.1	ECC Generation.....	684
8.3.3.2	ECC Generation for Partial Writes	685
8.3.3.3	ECC Checking	686
8.3.3.4	Scrubbing	690
8.3.3.4.1	ECC Example Using the H-Matrix.....	690
8.3.3.5	ECC Disabled	691
8.3.3.6	ECC Testing.....	691
8.3.4	Byte Parity Checking and Generation.....	692
8.3.4.1	Parity Generation	693
8.3.4.2	Parity Checking	694
8.3.4.3	Parity Disabled.....	694
8.3.4.4	Parity Testing	694
8.4	ECC Interrupts/Error Conditions.....	695
8.4.1	Single-Bit Error Detection	696
8.4.2	Multi-bit Error Detection	697
8.5	Parity Interrupts/Error Conditions	698
8.6	Register Definitions.....	699
8.6.1	SRAM Base Address Register — SRAMBAR.....	700
8.6.2	SRAM Upper Base Address Register — SRAMUBAR	700
8.6.3	SRAM ECC Control Register — SECR.....	701
8.6.4	SRAM ECC Log Register — SELOGR.....	702
8.6.5	SRAM ECC Address Register — SEAR.....	704



8.6.6	SRAM ECC Context Address Register — SECAR.....	704
8.6.7	SRAM ECC Test Register — SECTST	705
8.6.8	SRAM Parity Control and Status Register — SPARCSR.....	706
8.6.9	SRAM Parity Address Register — SPAR	707
8.6.10	SRAM Parity Upper Address Register — SPUAR.....	707
8.6.11	SRAM Memory Controller Interrupt Status Register — SMCISR	708
9.0	Peripheral Bus Interface Unit.....	709
9.1	Overview	710
9.2	Peripheral Bus Signals.....	711
9.2.1	Address Signal Definitions	711
9.2.2	Data Signal Definitions	711
9.2.3	Control/Status Signal Definitions	711
9.2.4	Bus Width	712
9.2.5	Detailed Signal Descriptions.....	713
9.2.6	Flash Memory Support	714
9.2.6.1	Flash Read Cycle	715
9.2.6.2	Flash Write Cycle	717
9.3	Register Definitions	718
9.3.1	PBI Control Register — PBCR	719
9.3.2	PBI Status Register — PBISR	719
9.3.3	Determining Block Sizes for Memory Windows	720
9.3.4	PBI Base Address Register 0 — PBBAR0	721
9.3.5	PBI Limit Register 0 — PBLR0	722
9.3.6	PBI Base Address Register 1 — PBBAR1	723
9.3.7	PBI Limit Register 1 — PBLR1	724
9.3.8	PBI Drive Strength Control Register — PBDSCR.....	725
9.3.9	Processor Frequency Register - PFR.....	726
9.3.10	External Strap Status Register 0 — ESSTR0	727
9.3.11	Unique ID Register 0 — UID0.....	728
9.3.12	Unique ID Register 1 — UID1.....	728
10.0	Exception Initiator and Boot Sequence	729
10.1	Overview	729
10.2	Theory of Operation.....	730
10.2.1	Core Identity	731
10.2.2	Targeted Reset	731
10.2.3	Inter-Processor Interrupt Generation	731
10.3	Intel XScale® Processor Boot Sequence	732
10.3.1	Theory of Operation.....	732
10.3.2	81341 and 81342 Boot Sequence	733
10.3.2.1	Boot Sequence — coreID0 and coreID1 simultaneous start running from Flash Memory.....	733
10.3.2.2	Boot Sequence — Both coreID0 and coreID1 run from SRAM Memory (no Flash Memory).....	733
10.3.2.3	Boot Sequence — coreID0 runs from Flash Memory and coreID1 runs from SRAM Memory.....	734
10.4	Register Definitions	735
10.4.1	Core Identification Register — CIDR.....	735
10.4.2	Reset Cause Status Register — RCSR.....	736
10.4.3	Software Interrupt Generation Register — SINTGENR.....	737
10.4.4	Targeted Reset Register — TARRSTR	738
11.0	Interrupt Controller Unit.....	739
11.1	Overview	739
11.2	Theory of Operation.....	740
11.2.1	Interrupt Controller Unit.....	740



11.3	The Intel XScale® Processor Exceptions Architecture	741
11.3.1	CPSR and SPSR	741
11.3.2	The Exception Process	742
11.3.3	Exception Priorities and Vectors	742
11.3.4	Software Requirements For Exception Handling	743
11.3.4.1	Nesting FIQ and IRQ Exceptions	743
11.4	Intel® 81341 and 81342 I/O Processors External Interrupt Interface	744
11.4.1	Interrupt Inputs	744
11.4.2	Outbound Interrupts	746
11.5	The Intel® 81341 and 81342 I/O Processors Interrupt Controller Unit	747
11.5.1	Programmer Model	748
11.5.1.1	Active Interrupt Source Control and Status	748
11.5.1.2	Prioritization and Vector Generation for Active Interrupt Sources ..	748
11.5.2	Operational Blocks	750
11.5.3	Intel® 81341 and 81342 I/O Processors: Internal Peripheral Interrupt	751
11.5.3.1	Normal Interrupt Sources	752
11.5.3.2	Error Interrupt Sources	753
11.5.4	High-Priority Interrupt (HPI#)	754
11.5.5	Timer Interrupts	754
11.5.6	Inter-Processor Interrupts	754
11.5.7	Intel XScale® processor Interrupts	754
11.5.8	Software Interrupts	754
11.6	Default Status	755
11.7	Interrupt Control Unit Registers	756
11.7.1	Interrupt Base Register — INTBASE	758
11.7.2	Interrupt Size Register — INTSIZE	759
11.7.3	IRQ Interrupt Vector Register — IINTVEC	760
11.7.4	FIQ Interrupt Vector Register — FINTVEC	761
11.7.5	Inter-Processor Interrupt Pending Register — IPIPNDR	762
11.7.6	Interrupt Pending Register 0 — INTPND0	763
11.7.7	Interrupt Pending Register 1 — INTPND1	764
11.7.8	Interrupt Pending Register 2 — INTPND2	765
11.7.9	Interrupt Pending Register 3 — INTPND3	766
11.7.10	Interrupt Control Register 0 — INTCTL0	767
11.7.11	Interrupt Control Register 1 — INTCTL1	770
11.7.12	Interrupt Control Register 2 — INTCTL2	772
11.7.13	Interrupt Control Register 3 — INTCTL3	773
11.7.14	Interrupt Steering Register 0 — INTSTR0	775
11.7.15	Interrupt Steering Register 1 — INTSTR1	778
11.7.16	Interrupt Steering Register 2 — INTSTR2	780
11.7.17	Interrupt Steering Register 3 — INTSTR3	781
11.7.18	IRQ Interrupt Source Register 0 — IINTSRC0	783
11.7.19	IRQ Interrupt Source Register 1 — IINTSRC1	786
11.7.20	IRQ Interrupt Source Register 2 — IINTSRC2	788
11.7.21	IRQ Interrupt Source Register 3 — IINTSRC3	789
11.7.22	FIQ Interrupt Source Register 0 — FINTSRC0	791
11.7.23	FIQ Interrupt Source Register 1 — FINTSRC1	794
11.7.24	FIQ Interrupt Source Register 2 — FINTSRC2	796
11.7.25	FIQ Interrupt Source Register 3 — FINTSRC3	797
11.7.26	Interrupt Priority Register 0 — IPR0	799
11.7.27	Interrupt Priority Register 1 — IPR1	800
11.7.28	Interrupt Priority Register 2 — IPR2	801
11.7.29	Interrupt Priority Register 3 — IPR3	802
11.7.30	Interrupt Priority Register 4 — IPR4	803
11.7.31	Interrupt Priority Register 5 — IPR5	804



11.7.32	Interrupt Priority Register 6 — IPR6	805
11.7.33	Interrupt Priority Register 7 — IPR7	806
12.0	Timers	807
12.1	Timer Operation	808
12.1.1	Basic Programmable Timer Operation	808
12.1.2	Watch Dog Timer Operation	809
12.1.3	Load/Store Access Latency for Timer Registers	810
12.2	Timer Interrupts	811
12.3	Timer State Diagram	812
12.4	Timer Registers	813
12.4.1	Power Up/Reset Initialization	813
12.4.2	Timer Mode Registers — TMR0:1	814
12.4.2.1	Bit 0 — Terminal Count Status Bit (TMRx.tc)	815
12.4.2.2	Bit 1 — Timer Enable (TMRx.enable)	815
12.4.2.3	Bit 2 — Timer Auto Reload Enable (TMRx.reload)	815
12.4.2.4	Bit 3 — Timer Register Privileged Read/Write Control (TMRx.pri) ...	816
12.4.2.5	Bits 4, 5 — Timer Input Clock Select (TMRx.csel1:0)	816
12.4.3	Timer Count Register — TCR0:1	817
12.4.4	Timer Reload Register — TRR0:1	817
12.4.5	Timer Interrupt Status Register — TISR	818
12.4.6	Watch Dog Timer Control Register — WDTCR	819
12.4.7	Watch Dog Timer Setup Register — WDTSR	819
12.5	Uncommon TCRX and TRRX Conditions	820
13.0	Inter-Processor Messaging Unit	821
13.1	Overview	821
13.2	Door Bell Registers	822
13.2.1	Door Bell Register Operation	823
13.3	Circular Queues	824
13.3.1	Circular Queue Operation	825
13.3.1.1	Send Queue Management	827
13.3.1.2	Receive Queue Management	827
13.4	Interrupts	828
13.4.1	Interrupt Prioritization	829
13.5	Power-up/Default Status	830
13.6	Register Definitions	830
13.6.1	Door Bell Control Register — DBCR	832
13.6.2	Door Bell Enable Register — DBER	834
13.6.3	Door Bell Assertion Register — DBAR	835
13.6.4	Door Bell Enable Other Processor Register — DBEOR	835
13.6.5	Send Queue Put/Get Pointer Register 0 — SQPG0	836
13.6.6	Send Queue Control Register 0 — SQCR0	837
13.6.7	Send Queue Lower Base Address Register 0 — SQLBAR0	838
13.6.8	Send Queue Upper Base Address Register 0 — SQUBAR0	838
13.6.9	Receive Queue Put/Get Pointer Register 0 — RQPG0	839
13.6.10	Receive Queue Control Register 0 — RQCR0	840
13.6.11	Receive Queue Lower Base Address Register 0 — RQLBAR0	840
13.6.12	Receive Queue Upper Base Address Register 0 — RQUBAR0	841
13.6.13	Send Queue Put/Get Pointer Register 1 — SQPG1	841
13.6.14	Send Queue Control Register 1 — SQCR1	842
13.6.15	Send Queue Lower Base Address Register 1 — SQLBAR1	843
13.6.16	Send Queue Upper Base Address Register 1 — SQUBAR1	843
13.6.17	Receive Queue Put/Get Pointer Register 1 — RQPG1	844
13.6.18	Receive Queue Control Register 1 — RQCR1	845
13.6.19	Receive Queue Lower Base Address Register 1 — RQLBAR1	846



13.6.20	Receive Queue Upper Base Address Register 1 — RQUBAR1	846
13.6.21	Send Queue Put/Get Pointer Register 2 — SQPG2	847
13.6.22	Send Queue Control Register 2 — SQCR2	848
13.6.23	Send Queue Lower Base Address Register 2 — SQLBAR2	849
13.6.24	Send Queue Upper Base Address Register 2 — SQUBAR2	849
13.6.25	Receive Queue Put/Get Pointer Register 2 — RQPG2	850
13.6.26	Receive Queue Control Register 2 — RQCR2	851
13.6.27	Receive Queue Lower Base Address Register 2 — RQLBAR2	852
13.6.28	Receive Queue Upper Base Address Register 2 — RQUBAR2	852
13.6.29	Send Queue Put/Get Pointer Register 3 — SQPG3	853
13.6.30	Send Queue Control Register 3 — SQCR3	854
13.6.31	Send Queue Lower Base Address Register 3 — SQLBAR3	854
13.6.32	Send Queue Upper Base Address Register 3 — SQUBAR3	855
13.6.33	Receive Queue Put/Get Pointer Register 3 — RQPG3	855
13.6.34	Receive Queue Control Register 3 — RQCR3	856
13.6.35	Receive Queue Lower Base Address Register 3 — RQLBAR3	856
13.6.36	Receive Queue Upper Base Address Register 3 — RQUBAR3	857
13.6.37	IMU Test and Set Registers — IMUTSR[0:511]	858
13.6.37.1	Endian Mode Support	859
14.0	SMBus Interface Unit	861
14.1	Overview	861
14.2	SMBus Interface	861
14.3	System Management Bus Interface	862
14.3.1	SMBus Controller	863
14.3.1.1	SMBus Commands	863
14.3.1.2	Initialization Sequence	864
14.3.2	SMBus Signaling	865
14.3.2.1	Overview	865
14.3.2.2	Waveforms	865
14.3.2.2.1	Start Phase	865
14.3.2.2.2	Stop Phase	866
14.3.2.2.3	ACK/NACK	866
14.3.2.2.4	Wait States	866
14.3.3	Architecture	867
14.3.3.1	Data Transfer Examples	869
14.3.3.2	Configuration and Memory Reads	869
14.3.3.3	Configuration and Memory Writes	872
14.3.4	Error Handling	874
14.3.5	SMBus Interface Reset	874
14.4	Register Definitions	875
14.4.1	SMBus Controller Command Register — SM_CMD	875
14.4.2	SMBus Controller Byte Count Register — SM_BC	876
14.4.3	SMBus Controller ADDR3 Register — SM_ADDR3	876
14.4.4	SMBus Controller ADDR2 Register — SM_ADDR2	876
14.4.5	SMBus Controller ADDR1 Register Number — SM_ADDR1	877
14.4.6	SMBus Controller ADDR0 Register Number — SM_ADDR0	877
14.4.7	SMBus Controller Data Register — SM_DATA	878
14.4.8	SMBus Controller Status Register — SM_STS	878
15.0	UARTs	879
15.1	Overview	879
15.1.1	Compatibility with 16550 and 16750	880
15.2	Signal Descriptions	881
15.3	Theory of Operation	882
15.3.1	FIFO Interrupt Mode Operation	883



15.3.1.1	Receiver Interrupt.....	883
15.3.1.2	Transmit Interrupt	883
15.3.2	Removing Trailing Bytes In Interrupt Mode	884
15.3.2.1	Character Time-out Interrupt	884
15.3.3	FIFO Polled Mode Operation	884
15.3.3.1	Receive Data Service	884
15.3.3.2	Transmit Data Service	884
15.3.4	Autoflow Control	885
15.3.4.1	RTS Autoflow	885
15.3.4.2	CTS Autoflow	885
15.3.5	Auto-Baud-Rate Detection	886
15.3.6	Manual Baud Rate Selection	887
15.4	Register Descriptions	888
15.4.1	UART x Receive Buffer Register	890
15.4.2	UART x Transmit Holding Register	890
15.4.3	UART x Interrupt Enable Register	891
15.4.4	UART x Interrupt Identification Register	892
15.4.5	UART x FIFO Control Register.....	894
15.4.6	UART x Line Control Register	896
15.4.7	UART x Modem Control Register	898
15.4.8	UART x Line Status Register	900
15.4.9	UART x Scratchpad Register	904
15.4.10	Divisor Latch Registers.....	905
15.4.11	UART x FIFO Occupancy Register.....	906
15.4.12	UART x Auto-Baud Control Register	907
15.4.13	UART x Auto-Baud Count Register	908
16.0	I ² C Bus Interface Units.....	909
16.1	Overview	909
16.2	Theory of Operation	910
16.2.1	Operational Blocks.....	912
16.2.2	I ² C Bus Interface Modes.....	914
16.2.3	Start and Stop Bus States	915
16.2.3.1	START Condition	916
16.2.3.2	No START or STOP Condition	916
16.2.3.3	STOP Condition	916
16.3	I ² C Bus Operation	917
16.3.1	Data and Addressing Management.....	917
16.3.1.1	Addressing a Slave Device	918
16.3.2	I ² C Acknowledge	919
16.3.3	Arbitration.....	920
16.3.3.1	SCL Arbitration.....	920
16.3.3.2	SDA Arbitration	921
16.3.4	Master Operations	922
16.3.5	Slave Operations	925
16.3.6	General Call Address.....	927
16.4	Slave Mode Programming Examples	928
16.4.1	Initialize Unit	928
16.4.2	Write 1 Byte as a Slave	928
16.4.3	Read 2 Bytes as a Slave	928
16.5	Master Programming Examples	929
16.5.1	Initialize Unit	929
16.5.2	Write 1 Byte as a Master	929
16.5.3	Read 1 Byte as a Master.....	929
16.5.4	Write 2 Bytes and Repeated Start Read 1 Byte as a Master	930
16.5.5	Read 2 Bytes as a Master — Send STOP Using the Abort.....	931



16.6	Glitch Suppression Logic	932
16.7	Reset Conditions	932
16.8	Register Definitions	932
16.8.1	I ² C Control Register x — ICRx	933
16.8.2	I ² C Status Register x — ISRx	935
16.8.3	I ² C Slave Address Register x — ISARx	937
16.8.4	I ² C Data Buffer Register x — IDBRx	938
16.8.5	I ² C Bus Monitor Register x — IBMRx	939
16.8.6	I ² C Manual Bus Control Register x — IMBCRx	940
17.0	General Purpose I/O Unit	941
17.1	General Purpose Input Output Support	941
17.1.1	General Purpose Inputs	941
17.1.2	General Purpose Outputs	941
17.1.3	Reset Initialization of General Purpose I/O Function	941
17.2	Register Definitions	942
17.2.1	GPIO Output Enable Register — GPOE	943
17.2.2	GPIO Input Data Register — GPID	944
17.2.3	GPIO Output Data Register — GPOD	946
18.0	PMON Unit	947
18.1	PMON Counters	947
18.2	Overview	947
18.2.1	Clock Counter Control	948
18.3	Definitions	949
18.4	Data Collection	950
18.4.1	Time Based Sampling	950
18.4.2	Hardware Event Based Control	952
18.4.3	Incrementing By More Than 1	954
18.4.4	Queue Analysis	955
18.5	Non-Register-Based Interfaces	961
18.5.1	Events Input Port	961
18.5.2	Output Signals	961
18.5.2.1	Indicator Output	962
18.5.2.2	Interrupt Output	962
18.5.3	Internal Bus Addresses	963
18.5.4	PMON Feature Enable Register - PMONEN	964
18.5.5	PMON Status Register - PMONSTAT	964
18.5.6	PMON Memory Mapped Registers	965
18.5.6.1	PMON Command Register 0-7 - PMON_CMD [0:7]	967
18.5.6.2	PMON Event Register 0-7 - PMON_EVR [0:7]	971
18.5.6.3	PMON Status Register 0-7 - PMON_STS [0:7]	972
18.5.6.4	PMON Data Register 0-7 - PMON_DATA [0:7]	974
18.5.7	PMON Events	975
18.5.7.1	Null Event	975
18.5.7.2	Clock Events	976
18.5.7.3	Threshold Events	976
18.5.7.4	DDR SDRAM Memory Controller Events	977
18.5.7.5	PCI Interface Events	981
18.5.7.6	PCI Express* Interface Events	982
18.5.7.7	North Internal Bus Events	983
18.5.7.8	South Internal Bus Events	984
19.0	Clocking and Reset	985
19.1	Clocking Overview	985
19.1.1	Clocking Theory of Operation	986
19.1.1.1	Clocking Region 1 (PCI Express)	986



19.1.1.2	Clocking Region 2 (PCI)	986
19.1.1.2.1	Central Resource Mode (PCIX_EP# = '1')	987
19.1.1.2.2	cPCI Hot-Swap Mode (PCIX_EP# = '0' and HS_SM# = '0')	988
19.1.1.2.3	End Point Mode (PCIX_EP# = 0 and HS_SM# = 1)	988
19.1.1.2.4	Secondary Clock Outputs	989
19.1.1.3	Clocking Region 3 (Internal Bus)	990
19.1.1.4	Clocking Region 4 (Peripheral Bus Interface)	990
19.1.1.5	Clocking Region 5	990
19.1.1.6	Clocking Region 7 (Intel XScale® Processor)	990
19.1.1.7	Clocking Region 8 (DDR SDRAM)	990
19.1.2	Clocking Region Summary	991
19.2	Reset Overview	993
19.2.1	Fundamental Reset	993
19.2.2	Software Reset	994
19.2.3	Secondary Bus Reset	994
19.2.4	PCI Reset	995
19.2.5	PCI Express* Hot Reset	995
19.2.6	WARM_RST# Reset Mechanism	995
19.2.7	Intel XScale® Processor Reset Mechanism	996
19.2.8	Internal Bus Reset	997
19.3	Reset Pins	1000
19.4	Device Function Select	1001
19.5	Reset Strapping Options	1002
20.0	Test Logic Unit and Testability	1005
20.1	Overview	1005
20.2	IEEE 1149.1 Standard Test Access Port (TAP)	1006
20.2.1	TAP Pin Description	1007
20.2.1.1	Test Clock (TCK)	1007
20.2.1.2	Test Mode Select (TMS)	1007
20.2.1.3	Test Data Input (TDI)	1007
20.2.1.4	Test Data Output (TDO)	1007
20.2.1.5	Asynchronous Reset (TRST#)	1007
20.2.2	TAP Controller	1008
20.2.2.1	Test-Logic-Reset State	1009
20.2.2.2	Run-Test/Idle State	1009
20.2.2.3	Select-DR-Scan State	1009
20.2.2.4	Capture-DR State	1009
20.2.2.5	Shift-DR State	1010
20.2.2.6	Exit1-DR State	1010
20.2.2.7	Pause-DR State	1010
20.2.2.8	Exit2-DR State	1010
20.2.2.9	Update-DR State	1010
20.2.2.10	Select-IR-Scan State	1011
20.2.2.11	Capture-IR State	1011
20.2.2.12	Shift-IR State	1011
20.2.2.13	Exit1-IR State	1011
20.2.2.14	Pause-IR State	1011
20.2.2.15	Exit2-IR State	1012
20.2.2.16	Update-IR State	1012
20.2.3	TAP Controller Registers	1013
20.2.3.1	Instruction Register	1013
20.2.3.2	Instructions	1014
20.2.3.3	Boundary-Scan Register	1015
20.2.3.4	Bypass Register	1015
20.2.3.5	Device Identification Register	1015
20.3	Definition of Terms	1016



21.0	Peripheral Registers	1017
21.1	Overview	1017
21.2	Accessing Peripheral Memory-Mapped Registers	1018
21.3	Accessing Peripheral Registers Using the Core Coprocessor Register Interface	1018
21.4	Architecturally Reserved Memory Space	1018
21.5	Default Memory Space Setup	1019
21.6	Peripheral Memory-Mapped Register Address Space	1022
21.6.1	Internal Units	1025
21.6.1.1	Application DMA 0-2	1025
21.6.1.2	Inter-Processor Messaging Unit	1027
21.6.1.3	SRAM Memory Controller	1029
21.6.1.4	Peripheral Bus Interface Unit	1030
21.6.1.5	System Controller	1030
21.6.1.6	Internal Bus Bridge	1031
21.6.1.7	DDR SDRAM Memory Controller	1032
21.6.1.8	I/O Pad Control	1033
21.6.1.9	UART 0-1	1036
21.6.1.10	GPIO	1037
21.6.1.11	I ² C Bus Interface Unit 0-2	1037
21.6.1.12	Messaging Unit	1038
21.6.1.13	PMON Unit	1040
21.6.2	Host Interface Units	1041
21.6.2.1	Address Translation Unit (PCI-X)	1042
21.6.2.2	Address Translation Unit (PCI-E)	1046
21.7	PCI Configuration Space	1051
21.8	Coprocessor Register Space	1052



Figures

1	Single-Core Intel® 81341 I/O Processor Functional Block Diagram.....	45
2	Dual-Core Intel® 81342 I/O Processor Functional Block Diagram	48
3	ATU Block Diagram	56
4	ATU Queue Architecture Block Diagram	57
5	Inbound Address Detection.....	62
6	Inbound Translation Example.....	63
7	4 Gbyte Section 0 of the Internal Bus Memory Map.....	73
8	Outbound Address Translation Windows	74
9	Inbound Byte Swapping for 32-bit PCI	83
10	Inbound Byte Swapping for 64-bit PCI	83
11	Outbound Byte Swapping for Transaction with Byte Count of 1	84
12	Outbound Byte Swapping for Transaction with Byte Count of 2.....	84
13	Outbound Byte Swapping for Transaction with Byte Count of 3 or Larger	84
14	PCI-X Initialization Pattern Setting / Drive	140
15	Intel® 81341 and 81342 I/O Processors PCI Interface Configuration Examples	141
16	ATU Interface Configuration Header Format.....	144
17	ATU Interface Extended Configuration Header Format (Power Management)	145
18	ATU Interface Extended Configuration Header Format (MSI-X Capability)	145
19	ATU Interface Extended Configuration Header Format (MSI Capability).....	146
20	ATU Interface Extended Configuration Header Format (PCI-X Capability Type 1).....	146
21	ATU Extended Configuration Header Format (Compact PCI Hot-Swap Capability)	147
22	ATU Interface Extended Configuration Header Format (VPD Capability)	147
23	ATU Block Diagram	236
24	ATU Queue Architecture Block Diagram	237
25	Inbound Address Detection.....	243
26	Inbound Translation Example.....	244
27	4 Gbyte Section 0 of the Internal Bus Memory Map.....	252
28	Outbound Address Translation Windows	254
29	Inbound Byte Swapping	259
30	Outbound Byte Swapping for Transaction with Byte Count of 1	260
31	Outbound Byte Swapping for Transactions with Byte Count of 2	260
32	Outbound Byte Swapping Transaction with Byte Count of 3 or Larger	260
33	Intel® 81341 and 81342 I/O Processors PCI Interface Configuration Examples	290
34	ATU Interface Configuration Header Format.....	293
35	ATU Interface Extended Configuration Header Format (Power Management)	294
36	ATU Interface Extended Configuration Header Format (MSI-X Capability)	294
37	ATU Interface Extended Configuration Header Format (MSI Capability).....	295
38	ATU Interface Extended Configuration Header Format (PCI Express Capability).....	295
39	ATU Interface Extended Configuration Header Format (VPD Capability)	295
40	PCI Express Vendor_Defined Message Header	389
41	PCI Memory Map.....	405
42	Internal Bus Memory Map	406
43	Overview of Circular Queue Operation.....	411
44	Circular Queue Operation	413
45	MSI-X Table and PBA Address Mapping Layout relative to the Host Interface	423
46	MSI-X Table and PBA Address Mapping Layout relative to the Internal Bus.....	424
47	Application DMA Channel Block Diagram	465
48	Basic Chain Descriptor Format	467
49	Full Chain Descriptor Format.....	469
50	P+Q Chain Descriptor Format (Minimum Size)	473
51	Dual XOR Descriptor Format.....	476
52	P+Q Update Descriptor Format	478
53	Chaining Operation	480



54	Synchronizing to Chained ADMA Operation.....	483
55	Optimization of an Unaligned ADMA Transfer.....	487
56	Optimization of an Unaligned ADMA Transfer.....	488
57	The Bit-wise XOR Algorithm.....	491
58	Hardware Assist XOR Unit.....	492
59	The P+Q RAID6 Mode Algorithm.....	494
60	Galois Field Logarithm Transformation Table.....	495
61	Galois Field Inverse Logarithm Transformation table.....	495
62	P+Q RAID-6 Generation Equation.....	496
63	The Bit-wise Dual XOR Algorithm.....	497
64	Hardware Assist XOR Unit.....	498
65	The P+Q Update Algorithm.....	501
66	An Example of Zero Result Buffer Check.....	502
67	Zero Result Buffer Check with P+Q RAID6.....	504
68	Example of a Memory Block Fill Operation.....	505
69	CRC-32C Generator Polynomial.....	508
70	Calculation of 32-bit CRC for iSCSI PDU.....	508
71	Application DMA Programming Model State Diagram.....	509
72	Pseudo Code: Application DMA Initialization.....	511
73	Pseudo Code: XOR Transfer Operation.....	511
74	Pseudo Code: Suspend Application DMA.....	512
75	Logical Data Access Paths with Parity Protection.....	513
76	Typical Internal Bus System Controller Block Diagram.....	539
77	Intel® 81341 and 81342 I/O Processors Block Diagram (External).....	541
78	Memory Controller Block Diagram.....	565
79	Intel® 81341 and 81342 I/O Processors Dual-Bank DDR SDRAM Memory Subsystem.....	574
80	DDR SDRAM 64-bit Memory Address Map.....	580
81	DDR SDRAM 64-bit Physical Map.....	580
82	Secondary DDR SDRAM Window Memory Map.....	582
83	64-bit to 32-bit Addressing.....	588
84	Page Hit/Miss Logic for 1Gbit Mode.....	590
85	Logical Memory Image of a DDR SDRAM Memory Subsystem.....	592
86	Supported DDR2 SDRAM Extended Mode Register Settings.....	594
87	Supported DDR SDRAM Mode Register Settings.....	595
88	DDR SDRAM Initialization Sequence (controlled with software).....	600
89	DMCU Active, Precharge, Refresh Command Timing Diagram.....	602
90	DDR SDRAM Pipelined Writes.....	605
91	DDR ECC Write Flow.....	608
92	Intel® 81341 and 81342 I/O Processors G-Matrix (generates the ECC).....	609
93	Sub 64-bit DDR SDRAM Write (D ₀).....	610
94	DDR ECC Read Data Flow.....	612
95	Intel® 81341 and 81342 I/O Processors H-Matrix (indicates the single-bit error location).....	613
96	Power Failure Sequence.....	618
97	Power Failure State Machine.....	619
98	Power Failure Sequence.....	620
99	DLLRCVREN Optimum Pulse Location.....	668
100	Intel® 81341 and 81342 I/O Processors SRAM Memory Controller Block Diagram.....	677
101	ECC Write Flow.....	684
102	81341 and 81342 G-Matrix (generates the ECC).....	685
103	ECC Read Data Flow.....	687
104	81341 and 81342 H-Matrix (indicates the single-bit error location).....	688
105	Logical Data Access Paths with Parity Protection.....	692
106	The Peripheral Bus Interface Unit.....	709
107	Data Width and Low Order Address Lines.....	712
108	Sixty-Four Mbyte Flash Memory System.....	714



109	120 ns Flash Single Transfer Read Cycle	715
110	120 ns Flash Burst Read Cycle	716
111	120 ns Flash Single Write Cycle ¹	717
112	Inter-Processor Communication Block Diagram	730
113	Interrupt Controller Block Diagram (Active Interrupt Source Registers)	748
114	Interrupt Controller Block Diagram (FIQ/IRQ Interrupt Vector Generation)	749
115	Intel® 81341 and 81342 I/O Processors Interrupt Controller Connections	750
116	Programmable Timer Functional Diagram	807
117	Timer Unit State Diagram	812
118	Door Bell Registers as Viewed by Processor 0 and Processor 1	823
119	Circular Queue Control Registers as Viewed from Each Processor	826
120	IMU handling Intel XScale® Processor running in Big Endian Mode	859
121	IMU handling Intel XScale® Processor running in Little Endian Mode	859
122	Basic SMBus Transfer Waveform	865
123	Start (S) / Repeat Start (Sr) Signaling	865
124	Stop (P) Signaling	866
125	ACK (A) Signaling	866
126	NACK (N) Signaling	866
127	DWORD Configuration Read Protocol (SMBus Block Write/Block Read, PEC Enabled)	869
128	DWORD Memory Read Protocol (SMBus Block Write/Block Read, PEC Enabled)	870
129	DWORD Configuration Read Protocol (SMBus Word Write/Word Read, PEC Enabled)	870
130	DWORD Configuration Read Protocol (SMBus Block Write/Block Read, PEC Disabled)	870
131	DWORD Memory Read Protocol (SMBus Block Write/Block Read, PEC Disabled)	870
132	DWORD Configuration Read Protocol (SMBus Word Write/Word Read, PEC Disabled)	871
133	DWORD Memory Read Protocol (SMBus Word Write/(Word, Byte) Read, PEC Enabled)	871
134	DWORD Memory Read Protocol (SMBus Word Write/Byte Read, PEC Enabled)	871
135	DWORD Configuration Write Protocol (SMBus Block Write, PEC Enabled)	872
136	DWORD Memory Write Protocol (SMBus Word Write, PEC Enabled)	872
137	DWORD Configuration Write Protocol (SMBus Byte Write, PEC Enabled)	873
138	Example UART Data Frame	882
139	NRZ Bit Encoding Example – (0100 1011)	882
140	I ² C Bus Configuration Example	910
141	I ² C Bus Interface Unit Block Diagram	912
142	Start and Stop Conditions	915
143	START and STOP Conditions	916
144	Data Format of First Byte in Master Transaction	918
145	Acknowledge on the I ² C Bus	919
146	Clock Synchronization During the Arbitration Procedure	920
147	Arbitration Procedure of Two Masters	921
148	Master-Receiver Read from Slave-Transmitter	924
149	Master-Receiver Read from Slave-Transmitter / Repeated Start / Master-Transmitter Write to Slave-Receiver ⁹²⁴	
150	A Complete Data Transfer	924
151	Master-Transmitter Write to Slave-Receiver	926
152	Master-Receiver Read to Slave-Transmitter	926
153	Master-Receiver Read to Slave-Transmitter, Repeated START, Master-Transmitter Write to Slave-Receiver ⁹²⁶	
154	General Call Address	927
155	Example Block Diagram of Component with Counter	947
156	Conceptual Diagram of Counter Array	948
157	Example Block Diagram of Single PMON Counter	949
158	Flowchart of Example Commands Sequence	950
159	Block Diagram and Waveforms of Time Based Sampling Example	951
160	Block Diagram and Waveforms of Time Based Sampling Example	953
161	Block Diagram & Waveforms of Time Based Sampling Example	954



162 Block Diagram & Waveforms of Time Based Sampling Example.....	955
163 Block Diagram of HOQ Histogram Example.....	958
164 Waveforms of HOQ Histogram Example	959
165 Processing of HOQ Histogram Example	960
166 Output from HOQ Histogram Example	960
167 Indicator Tree	962
168 Intel® 81341 and 81342 I/O Processors Clocking Regions Diagram	985
169 IEEE 1149.1 Std. Block Diagram.....	1006
170 Timing of Actions in a TAP Controller State.....	1008
171 TAP Controller State Diagram.....	1008
172 IOP Device ID Register.....	1015
173 Memory Address Space	1020
174 Intel® 81341 and 81342 I/O Processors Memory Address Space	1021



Tables

1	ATU Command Support.....	59
2	Outbound Address Translation Control	73
3	Internal Bus-to-PCI Command Translation for Memory Windows.....	75
4	Internal Bus-to-PCI Command Translation for I/O Window	75
5	Compact PCI Hot-Swap.....	85
6	HS_FREQ Encoding	86
7	Inbound Queues	88
8	Inbound Read Prefetch Data Sizes	89
9	PCI to Internal Bus Command Translation for All Inbound Transactions	90
10	Outbound Queues	91
11	ATU Inbound Data Flow Ordering Rules	92
12	ATU Outbound Data Flow Ordering Rules	93
13	Inbound Transaction Ordering Summary	95
14	Outbound Transaction Ordering Summary	96
15	Parity Generation	97
16	ATU Error Reporting Summary - PCI Interface	124
17	ATU Error Reporting Summary - Internal Bus Interface	129
18	CR_FREQ[1:0] Encoding	136
19	Device Mode/Frequency Capability Reporting	137
20	PCI Bus Frequency Initialization	138
21	PCI-X Initialization Pattern	139
22	Address Translation Unit Registers	148
24	PCI-X Pad Registers	151
23	ATU Internal Bus Memory Mapped Register Range Offsets	151
25	ATU Vendor ID Register - ATUVID	152
26	ATU Device ID Register - ATUDID.....	152
27	ATU Command Register - ATUCMD	153
28	ATU Status Register - ATUSR.....	154
29	ATU Revision ID Register - ATURID	156
30	ATU Class Code Register - ATUCCR.....	156
31	ATU Cacheline Size Register - ATUCLSR	157
32	ATU Latency Timer Register - ATULT.....	157
33	ATU Header Type Register - ATUHTR	158
34	ATU BIST Register - ATUBISTR	159
35	Inbound ATU Base Address Register 0 - IABAR0	160
36	Inbound ATU Upper Base Address Register 0 - IAUBAR0.....	161
37	Inbound ATU Base Address Register 1 - IABAR1	162
38	Inbound ATU Upper Base Address Register 1 - IAUBAR1	163
39	Inbound ATU Base Address Register 2 - IABAR2.....	164
40	Inbound ATU Upper Base Address Register 2 - IAUBAR2.....	165
41	ATU Subsystem Vendor ID Register - ASVIR.....	166
42	ATU Subsystem ID Register - ASIR.....	166
43	Expansion ROM Base Address Register -ERBAR	167
44	ATU Capabilities Pointer Register - ATU_Cap_Ptr	168
45	Memory Block Size Read Response	169
46	ATU Base Registers and Associated Limit Registers	170
47	ATU Interrupt Line Register - ATUILR.....	171
48	ATU Interrupt Pin Register - ATUIPR.....	172
49	ATU Minimum Grant Register - ATUMGNT	172
50	ATU Maximum Latency Register - ATUMLAT	173
51	Inbound ATU Limit Register 0 - IALR0	174
52	Inbound ATU Translate Value Register 0 - IATVR0	175
53	Inbound ATU Upper Translate Value Register 0 - IAUTVR0	175



54	Inbound ATU Limit Register 1 - IALR1	176
55	Inbound ATU Translate Value Register 1 - IATVR1.....	177
56	Inbound ATU Upper Translate Value Register 1 - IAUTVR1	177
57	Inbound ATU Limit Register 2 - IALR2	178
58	Inbound ATU Translate Value Register 2 - IATVR2.....	179
59	Inbound ATU Upper Translate Value Register 2 - IAUTVR2	179
60	Expansion ROM Limit Register - ERLR.....	180
61	Expansion ROM Translate Value Register - ERTVR.....	181
62	Expansion ROM Upper Translate Value Register - ERUTVR.....	181
63	ATU Configuration Register - ATUCR	182
64	PCI Configuration and Status Register - PCSR	183
65	ATU Interrupt Status Register - ATUISR.....	186
66	ATU Interrupt Mask Register - ATUIMR.....	188
67	VPD Capability Identifier Register - VPD_Cap_ID	190
68	VPD Next Item Pointer Register - VPD_Next_Item_Ptr	190
69	VPD Address Register - VPDAR.....	191
70	VPD Data Register - VPDDR.....	191
71	PM_Capability Identifier Register - PM_Cap_ID	192
72	PM Next Item Pointer Register - PM_Next_Item_Ptr	192
73	ATU Power Management Capabilities Register - APMCR.....	193
74	ATU Power Management Control/Status Register - APMCSR.....	194
75	Scratch Pad Register - ATUSPR.....	195
76	PCI-X_Capability Identifier Register - PCI-X_Cap_ID	195
77	PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr.....	196
78	PCI-X Command Register - PCIXCMD	196
79	PCI-X Status Register - PCIXSR.....	198
80	ECC Control and Status Register - ECCCSR	200
81	ECC First Address Register - ECCFAR.....	203
82	ECC Second Address Register - ECCSAR	204
83	ECC Attribute Register - ECCAR.....	205
84	HS_CAPID - Hot-Swap Cap ID.....	205
85	HS_NXTP - Next Item Pointer.....	206
86	HS_CNTRL - Hot-Swap Control/Status Register.....	207
87	Inbound ATU Base Address Register 3 - IABAR3	209
88	Inbound ATU Upper Base Address Register 3 - IAUBAR3	210
89	Inbound ATU Limit Register 3 - IALR3	211
90	Inbound ATU Translate Value Register 3 - IATVR3.....	212
91	Inbound ATU Upper Translate Value Register 3 - IAUTVR3	212
92	Outbound I/O Base Address Register - OIOBAR	213
93	Outbound I/O Window Translate Value Register - OIOWTVR.....	214
94	Outbound Upper Memory Window Base Address Register 0 - OUMBAR0	215
95	Outbound Upper 32-bit Memory Window Translate Value Register 0- OUMWTVR0.....	216
96	Outbound Upper Memory Window Base Address Register 1 - OUMBAR1	217
97	Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1.....	218
98	Outbound Upper Memory Window Base Address Register 2- OUMBAR2	219
99	Outbound Upper 32-bit Memory Window Translate Value Register 2- OUMWTVR2.....	220
100	Outbound Upper Memory Window Base Address Register 3 - OUMBAR3	221
101	Outbound Upper 32-bit Memory Window Translate Value Register 3- OUMWTVR3.....	222
102	Outbound Configuration Cycle Address Register - OCCAR.....	223
103	Outbound Configuration Cycle Data Register - OCCDR.....	224
104	Outbound Configuration Cycle Function Number Register - OCCFN	224
105	PCI Interface Error Control and Status Register - PIECSR.....	225
106	PCI Interface Error Address Register - PCIEAR.....	226
107	PCI Interface Error Upper Address Register - PCIEUAR.....	227
108	PCI Interface Error Context Address Register - PCIECAR	228



109	Internal Arbiter Control Register - IACR	229
110	Multi-Transaction Timer - MTT	230
111	PCIX RCOMP Control Register - PRCR.....	231
112	PCIX Pad ODT Drive Strength Manual Override Values Registers - PPODSMOVR.....	232
113	PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3V/1.5V Switch Supply Voltage) — PPDSMOVR3.3_1.5	233
114	PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V Dedicated Supply Voltage) — PPDSMOVR3.3	234
115	Supported Address Spaces and Transaction Types	240
116	ATU Command Support.....	241
117	Inbound Vendor_Defined Message Type 0 Response.	248
118	Outbound Address Translation Control	250
119	Internal Bus-to-PCI Command Translation for Memory Windows	251
120	Internal Bus-to-PCI Command Translation for I/O Window	251
121	Supported Message Types	262
122	Inbound Queues	265
123	PCI to Internal Bus Command Translation for All Inbound Transactions	266
124	Outbound Queues	267
125	ATU Inbound Data Flow Ordering Rules	268
126	ATU Outbound Data Flow Ordering Rules	269
127	Inbound Transaction Ordering Summary	271
128	Outbound Transaction Ordering Summary	271
129	Parity Generation	272
130	Advisory Error Cases	275
131	PCI Express Error Summary	280
133	Internal Bus Error Summary	282
132	Root Complex Error Summary	282
134	ATU Internal Bus Memory Mapped Register Range Offsets	297
135	ATU PCI Configuration Register Space	297
136	ATU Vendor ID Register - ATUVID	301
137	ATU Device ID Register - ATUDID	301
138	ATU Command Register - ATUCMD	302
139	ATU Status Register - ATUSR.....	303
140	ATU Revision ID Register - ATURID	304
141	ATU Class Code Register - ATUCCR.....	304
142	ATU Cacheline Size Register - ATUCLSR	305
143	ATU Latency Timer Register - ATULT	305
144	ATU Header Type Register - ATUHTR	306
145	ATU BIST Register - ATUBISTR	307
146	Inbound ATU Base Address Register 0 - IABAR0	308
147	Inbound ATU Upper Base Address Register 0 - IAUBAR0.....	309
148	Memory Block Size Read Response	310
149	ATU Base Registers and Associated Limit Registers	311
150	Inbound ATU Base Address Register 1 - IABAR1	312
151	Inbound ATU Upper Base Address Register 1 - IAUBAR1.....	313
152	Inbound ATU Base Address Register 2 - IABAR2.....	314
153	Inbound ATU Upper Base Address Register 2 - IAUBAR2.....	315
154	ATU Subsystem Vendor ID Register - ASVIR	316
155	ATU Subsystem ID Register - ASIR.....	316
156	Expansion ROM Base Address Register -ERBAR	317
157	ATU Capabilities Pointer Register - ATU_Cap_Ptr	318
158	ATU Interrupt Line Register - ATUILR.....	319
159	ATU Interrupt Pin Register - ATUIPR	320
160	ATU Minimum Grant Register - ATUMGNT.....	320
161	ATU Maximum Latency Register - ATUMLAT.....	321



162 Inbound ATU Limit Register 0 - IALR0	322
163 Inbound ATU Translate Value Register 0 - IATVR0.....	323
164 Inbound ATU Upper Translate Value Register 0 - IAUTVR0	323
165 Inbound ATU Limit Register 1 - IALR1	324
166 Inbound ATU Translate Value Register 1 - IATVR1.....	325
167 Inbound ATU Upper Translate Value Register 1 - IAUTVR1	325
168 Inbound ATU Limit Register 2 - IALR2	326
169 Inbound ATU Translate Value Register 2 - IATVR2.....	327
170 Inbound ATU Upper Translate Value Register 2 - IAUTVR2	328
171 Expansion ROM Limit Register - ERLR.....	328
172 Expansion ROM Translate Value Register - ERTVR	329
173 Expansion ROM Upper Translate Value Register - ERUTVR.....	329
174 ATU Configuration Register - ATUCR	330
175 PCI Configuration and Status Register - PCSR	331
176 ATU Interrupt Status Register - ATUISR.....	333
177 ATU Interrupt Mask Register - ATUIMR.....	336
178 PCI Express Message Control and Status Register - PEMCSR	337
179 PCI Express Link Control and Status Register - PELCSR	338
180 VPD Capability Identifier Register - VPD_Cap_ID	339
181 VPD Next Item Pointer Register - VPD_Next_Item_Ptr	339
182 VPD Address Register - VPDAR.....	340
183 VPD Data Register - VPDDR.....	340
184 PM_Capability Identifier Register - PM_Cap_ID	341
185 PM Next Item Pointer Register - PM_Next_Item_Ptr	341
186 ATU Power Management Capabilities Register - APMCR.....	342
187 ATU Power Management Control/Status Register - APMCSR.....	343
188 Scratch Pad Register - ATUSPR.....	344
189 PCI Express Capability Identifier Register - PCIE_CAPID	344
190 PCI Express Next Item Pointer Register - PCIE_NXTP	345
191 PCI Express Capabilities Register PCIE_CAP	346
192 PCI Express Device Capabilities Register - PCIE_DCAP	347
193 PCI Express Device Control Register - PE_DCTL.....	348
194 PCI Express Device Status Register PE_DSTS	350
195 PCI Express Link Capabilities Register - PE_LCAP	351
196 PCI Express Link Control Register PE_LCTL	352
197 PCI Express Link Status Register PE_LSTS	353
198 PCI Express Slot Capabilities Register - PE_SCAP	354
199 PCI Express Slot Control Register PE_SCR.....	355
200 PCI Express Slot Status Register PE_SSTS	356
201 PCI Express Root Control Register - PE_RCR	357
202 PCI Express Root Status Register PE_RSR.....	358
203 PCI Express Advanced Error Capability Identifier - ADVERR_CAPID.....	359
204 PCI Express Uncorrectable Error Status - ERRUNC_STS	360
205 PCI Express Uncorrectable Error Mask - ERRUNC_MSK.....	361
206 PCI Express Uncorrectable Error Severity - ERRUNC_SEV.....	362
207 PCI Express Correctable Error Status - ERRCOR_STS.....	363
208 PCI Express Correctable Error Mask - ERRCOR_MSK.....	364
209 Advanced Error Control and Capability Register - ADVERR_CTL.....	365
210 PCI Express Advanced Error Header Log - ADVERR_LOG0	365
211 PCI Express Advanced Error Header Log - ADVERR_LOG1	366
212 PCI Express Advanced Error Header Log - ADVERR_LOG2	366
213 PCI Express Advanced Error Header Log - ADVERR_LOG3	367
214 Root Error Command Register - RERR_CMD	367
215 Root Error Status Register - RERR_SR.....	368
216 Error Source Identification Register RERR_ID	369



217	Device Serial Number Capability - DSN_CAP	369
218	Device Serial Number Lower DW Register - DSN_LDW	370
219	Device Serial Number Upper DW Register - DSN_UDW	370
220	PCI Express Advisory Error Control Register PIE_AEC	371
221	Power Budgeting Enhanced Capability Header - PWRBGT_CAPID	372
222	Power Budgeting Data Select Register - PWRBGT_DSEL	372
223	Power Budgeting Data Register - PWRBGT_DATA	373
224	Power Budgeting Capability Register - PWRBGT_CAP	374
225	Power Budgeting Information Registers[0:23]—PWRBGT_INFO[0:23]	375
226	Outbound I/O Base Address Register - OIOBAR	376
227	Outbound I/O Window Translate Value Register - OIOWTVR	377
228	Outbound Upper Memory Window Base Address Register 0 - OUMBAR0	378
229	Outbound Upper 32-bit Memory Window Translate Value Register 0- OUMWTVR0	379
230	Outbound Upper Memory Window Base Address Register 1 - OUMBAR1	380
231	Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1	381
232	Outbound Upper Memory Window Base Address Register 2- OUMBAR2	382
233	Outbound Upper 32-bit Memory Window Translate Value Register 2- OUMWTVR2	383
234	Outbound Upper Memory Window Base Address Register 3 - OUMBAR3	384
235	Outbound Upper 32-bit Memory Window Translate Value Register 3- OUMWTVR3	385
236	Outbound Configuration Cycle Address Register - OCCAR	386
237	Outbound Configuration Cycle Data Register - OCCDR	387
238	Outbound Configuration Cycle Function Number - OCCFN	388
239	Inbound Vendor Defined Message Header Register0 - IVMHR0	389
240	Inbound Vendor Defined Message Header Register 1 - IVMHR1	390
241	Inbound Vendor Defined Message Header Register 2 - IVMHR2	391
242	Inbound Vendor Defined Message Header Register 3 - IVMHR3	392
243	Inbound Vendor Defined Message Payload Register - IVMPR	392
244	Outbound Vendor Defined Message Header Register0 - OVMHR0	393
245	Outbound Vendor Defined Message Header Register 1 - OVMHR1	394
246	Outbound Vendor Defined Message Header Register 2 - OVMHR2	395
247	Outbound Vendor Defined Message Header Register 3 - OVMHR3	395
248	Outbound Vendor Defined Message Payload Register - OVMPR	396
249	PCI Interface Error Control and Status Register - PIE_CSR	397
250	PCI Interface Error Status - PIE_STS	398
251	PCI Interface Error Mask - PIE_MSK	399
252	PCI Interface Error Header Log - PIE_LOG0	400
253	PCI Interface Error Header Log 1 - PIE_LOG1	400
254	PCI Interface Error Header Log 2 - PIE_LOG2	401
255	PCI Interface Error Header Log - PIE_LOG3	401
256	PCI Interface Error Descriptor Log - PIE_DLOG	402
257	ATU Reset Control Register - ATURCR	402
258	MU Summary	406
259	Circular Queue Ordering Requirements	407
260	Circular Queue Summary	410
261	Queue Starting Addresses	412
262	Circular Queue Summary	418
263	Circular Queue Status Summary	418
264	Message Unit Registers	426
265	Inbound Message Register - IMRx	427
266	Outbound Message Register - OMRx	427
267	Inbound Doorbell Register - IDR	428
268	Inbound Interrupt Status Register - IISR	429
269	Inbound Interrupt Mask Register - IIMR	430
270	Outbound Doorbell Register - ODR	431
271	Outbound Interrupt Status Register - OISR	432



272 Outbound Interrupt Mask Register - OIMR	433
273 Inbound Reset Control and Status Register - IRCSR	434
274 Outbound Reset Control and Status Register - ORCSR	435
275 MSI Inbound Message Register - MIMR	436
276 MU Configuration Register - MUCR	437
277 Queue Base Address Register - QBAR	438
278 Inbound Free Head Pointer Register - IFHPR	439
279 Inbound Free Tail Pointer Register - IFTPR	439
280 Inbound Post Head Pointer Register - IPHPR	440
281 Inbound Post Tail Pointer Register - IPTPR	440
282 Outbound Free Head Pointer Register - OFHPR	441
283 Outbound Free Tail Pointer Register - OFTPR	441
284 Outbound Post Head Pointer Register - OPHPR	442
285 Outbound Post Tail Pointer Register - OPTPR	442
286 Index Address Register - IAR	443
287 MU Base Address Register - MUBAR	444
288 MU Upper Base Address Register - MUUBAR	445
289 MU MSI-X Table Message Address Registers - M_MT_MAR [0:7]	446
290 MU MSI-X Table Message Upper Address Registers - M_MT_MUAR [0:7]	447
291 MU MSI-X Table Message Upper Address Registers - M_MT_MUAR [0:7]	448
292 MU MSI-X Table Message Vector Control Registers - M_MT_MVCR [0:7]	449
293 MU MSI-X Pending Bits Array Register - M_MPBAR	450
294 MSI Capability Identifier Register - MSI_Cap_ID	450
295 MSI Next Item Pointer Register - MSI_Next_Ptr	451
296 Message Control Register - Message_Control	452
297 Message Address Register - Message_Address	453
298 Message Upper Address Register - Message_Upper_Address	454
299 Message Data Register - Message_Data	455
300 MSI-X_Capability Identifier Register - MSI-X_Cap_ID	456
301 MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr	457
302 MSI-X Message Control Register - MSI-X_MCR	458
303 MSI-X Table Offset Register - MSI-X_Table_Offset	459
304 MSI-X Pending Bit Array Offset Register - MSI-X_PBA Offset	460
305 MU MSI-X Control Register X — MMCRx	461
306 Inbound MSI Interrupt Pending Registers — IMIPR [0:3]	462
307 Basic and Full Chain Descriptor Functions Support	466
308 ADMA Operation Data Transfer Options	486
309 Supported ADMA Operations per ADMA Channel	490
310 Multiple Descriptor Usage for CRC Calculation of a 3 Fragment PDU	507
311 Address Parity Checking/Generation	515
312 Data Parity Checking/Generation	515
313 ADMA Interrupt Summary	518
314 ATU Internal Bus Memory Mapped Register Range Offsets	520
315 Application DMA Unit Registers	520
316 ADMA Channel Control Register x — ACCRx	521
317 ADMA Channel Status Register x — ACSRx	522
318 ADMA Descriptor Address Register x — ADARx	524
319 Internal Interface Parity Control Register x — IIPCR x	524
320 ADMA Next Descriptor Address Register x — ANDARx	525
321 ADMA Descriptor Control Register x — ADCRx	526
322 CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx	529
323 ADMA Byte Count Register x — ABCRx	530
324 Destination Lower Address Register / P_Destination Lower Address x — DLADRx	531



325 Destination Upper Address / PQ_Destination Upper Address Register x — DUADRx	532
326 Source Lower Address Register 0...15_x — SLAR0...15_x	534
327 Source Upper Address Register 0...15_x — SUAR0...15_x	536
328 Intel® 81341 and 81342 I/O Processors Initiator IDs	538
329 Address and Data Parity Testing Initiator IDs	540
330 Data Parity Testing Completer IDs	540
331 Bridge supported Internal Bus Commands	543
332 Ordering and Passing Rules for both Inbound and Outbound Transactions	545
333 Internal Bus Arbitration Control Register — IBACR	551
334 South Internal Bus Address Test Control Register — SIBATCR	553
335 South Internal Bus Data Test Control Register — SIBDTCR	554
336 Peripheral Memory-Mapped Register Base Address Register — PMMRBAR	555
337 Memory Block Size Limit Register Value	556
338 Bridge Window Base Address Register — BWBAR	557
339 Bridge Window Upper Base Address Register — BWUBAR	558
340 Bridge Limit Register — BWLR	559
341 Bridge Error Control and Status Register — BECSR	560
342 Bridge Error Address Register — BERAR	562
343 Bridge Error Upper Address Register — BERUAR	562
344 Commonly Used Terms	564
345 DDR SDRAM Memory Configuration Options	572
346 DDR2 SDRAM Interface Signals	573
347 DDR2 SDRAM Address Translation for 512 Mbit (x16) and 1 Gbit (x16) Devices (SDCR0[6] cleared) 575	
348 DDR2 SDRAM Address Translation for 512 Mbit (x8), and 1 Gbit (x8) Devices (SDCR0[6] cleared) 575	
349 DDR2 SDRAM Address Translation for 2 Gbit (x16) Device (SDCR0[6] set)	575
350 DDR2 SDRAM Address Translation for 2 Gbit (x8) Device (SDCR0[6] set)	575
351 DDR2 SDRAM Address Translation for 512 Mbit (x16) Device (SDCR0[6] set)	576
352 DDR2 SDRAM Address Translation for 1 Gbit (x16) Device (SDCR0[6] set)	576
353 DDR2 SDRAM Address Translation for 512 Mbit (x8) Device (SDCR0[6] set)	576
354 DDR2 SDRAM Address Translation for 1 Gbit (x8) Device (SDCR0[6] set)	576
355 DDR2 SDRAM Address Translation for 2 Gbit (x16) Device (SDCR0[6] set)	576
356 DDR2 SDRAM Address Translation for 2 Gbit (x8) Device (SDCR0[6] set)	577
357 Supported DDR2 SDRAM Configurations (SDCR0[6] cleared)	578
358 Supported DDR2 SDRAM Configurations (SDCR0[6] set)	578
359 DDR SDRAM Address Register Summary	579
360 Programming Values for the DDR SDRAM Bank Size Register	584
361 Programming Values for the DDR SDRAM Bank Number	584
362 Programming Values for the Secondary DDR SDRAM Window Size	584
363 Programming Values for the DDR SDRAM 32-bit Size Register (S32SR[29:20])	584
364 DDR SDRAM Commands	593
365 SDIR Encoding Examples	596
367 SDRAM Mode Register (MRS) Definition	599
366 SDRAM Extended Mode Register (EMRS) Definition	599
368 SDCR[1:0] Timing Parameters Summary	601
369 Typical Refresh Frequency Register Values	606
370 Syndrome Decoding	611
371 DMCU Error Response	621
372 Memory Controller Registers	625
373 Memory Controller Pad Registers	626
374 DDR SDRAM Initialization Register — SDIR	627
375 DDR SDRAM Control Register 0 — SDCR0	628
376 DDR SDRAM Control Register 1 - SDCR1	630



377 SDRAM Base Register — SDBR	632
378 SDRAM Upper Base Register — SDUBR	633
379 Secondary SDRAM Base Register - SSDBR	634
380 SDRAM Bank Size Register — SBSR	635
381 DDR SDRAM 32-bit Region Size Register — S32SR	637
382 DDR ECC Control Register — DECCR	638
383 DDR ECC Log Registers — DELOG0, DELOG1	639
384 DDR ECC Address Registers — DEAR0, DEAR1	641
385 DDR ECC Context Address Registers — DECAR 0, DECAR 1	642
386 DDR ECC Context Upper Address Registers — DECUAR 0, DECUAR 1	643
387 DDR ECC Test Register — DECTST	643
388 DDR Parity Control and Status Register — DPCSR	644
389 DDR Parity Address Register — DPAR	645
390 DDR Parity Upper Address Register — DPUAR	645
391 DDR Parity Context Address Register — DPCAR	646
392 DDR Parity Context Upper Address Register — DPCUAR	646
393 DDR Memory Controller Interrupt Status Register — DMCISR	647
394 DMCU Port Transaction Count Register — DMPTCR	648
395 DMCU Preemption Control Register - DMPCR	650
396 Refresh Frequency Register — RFR	651
397 DDR RCOMP Control Register — DRCSR	652
398 RCOMP Pad Drive Strength Select — RPDSR	654
399 DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR	655
400 DQ Pad Drive Strength Manual Override Values Register — DQPDSR	656
401 MA Pad Drive Strength Manual Override Values Register — ADPDSR	657
402 MCLK Pad Drive Strength Manual Override Values Register — MPDSR	658
403 CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR	659
404 DLL Delay Register 0 — DLLR0	660
405 DLL Delay Register 1 — DLLR1	661
406 DLL Delay Register 2 — DLLR2	662
407 DLL Delay Register 3 — DLLR3	663
408 DLL Delay Register 4 — DLLR4	664
409 DLL Delay for Receive Enable Register — DLLRCVER	665
410 DDR RCOMP Status Register for 15 Ohm RCOMP — DRSR15	672
411 DDR RCOMP Status Register for 17.9 Ohm RCOMP — DRSR17	672
412 DDR RCOMP Status Register for 25 Ohm RCOMP — DRSR25	673
413 DDR RCOMP Status Register for 35 Ohm RCOMP — DRSR35	673
414 DDR RCOMP Status Register for 50 Ohm RCOMP — DRSR50	674
415 Commonly Used Terms	676
416 Syndrome Decoding	686
417 Data Parity Checking/Generation	693
418 SMCU Error Response	695
419 Memory Controller Register	699
420 SRAM Base Address Register — SRAMBAR	700
421 SRAM Upper Base Address Register — SRAMUBAR	700
422 SRAM ECC Control Register — SECR	701
423 SRAM ECC Log Register — SELOG	702
424 SRAM ECC Address Register — SEAR	704
425 SRAM ECC Context Address Register — SECAR	704
426 SRAM ECC Test Register — SECTST	705
427 SRAM Parity Control and Status Register — SPARCSR	706
428 SRAM Parity Address Registers — SPAR	707
429 SRAM Parity Upper Address Register — SPUAR	707
430 SRAM Memory Controller Interrupt Status Register — SMCISR	708
431 Bus Signal Descriptions	713



432 Flash Wait State Profile Programming ¹	716
433 Peripheral Bus Interface Registers	718
434 PBI Control Register — PBCR	719
435 PBI Status Register — PBISR	719
436 Memory Block Size Limit Register Values	720
437 PBI Base Address Register 0 — PBBAR0	721
438 PBI Limit Register 0 — PBLR0	722
439 PBI Base Address Register 1 — PBBAR1	723
440 PBI Limit Register 1 — PBLR1	724
441 PBI Drive Strength Control Register — PBDSCR	725
442 Processor Frequency Register - PFR	726
443 External Strap Status Register 0 — ESSTS0	727
444 Unique ID Register 0 — UID0	728
445 Unique ID Register 1 — UID1	728
446 Exception Initiator Register Addresses	735
447 Core Identification Register — CIDR	735
448 Reset Cause Status Register — RCSR	736
449 Software Interrupt Generation Register — SINTGENR	737
450 Targeted Reset Register — TARRSTR	738
451 Exception Priorities And Vectors	742
452 Interrupt Input Pin Descriptions	745
453 Interrupt Output Pin Descriptions	746
454 Normal Interrupt Sources	752
455 Error Interrupt Sources	753
456 Default Interrupt Routing and Status Values	755
457 Interrupt Controller Co-Processor Register Addresses	756
458 Interrupt Base Register — INTBASE	758
459 Interrupt Size Register — INTSIZE	759
460 IRQ Interrupt Vector Register- IINTVEC	760
461 FIQ Interrupt Vector Register- FINTVEC	761
462 Inter-Processor Interrupt Pending Register — IPIPNDR	762
463 Interrupt Pending Register 0 — INTPND0	763
464 Interrupt Pending Register 1 — INTPND1	764
465 Interrupt Pending Register 2 — INTPND2	765
466 Interrupt Pending Register 3 — INTPND3	766
467 Interrupt Control Register 0 — INTCTL0	767
468 Interrupt Control Register 1 — INTCTL1	770
469 Interrupt Control Register 2 — INTCTL2	772
470 Interrupt Control Register 3 — INTCTL3	773
471 Interrupt Steering Register 0 — INTSTR0	775
472 Interrupt Steering Register 1 — INTSTR1	778
473 Interrupt Steering Register 2 — INTSTR2	780
474 Interrupt Steering Register 3 — INTSTR3	781
475 IRQ Interrupt Source Register 0 — IINTSRC0	783
476 IRQ Interrupt Source Register 1 — IINTSRC1	786
477 IRQ Interrupt Source Register 2 — IINTSRC2	788
478 IRQ Interrupt Source Register 3 — IINTSRC3	789
479 FIQ Interrupt Source Register 0 — FINTSRC0	791
480 FIQ Interrupt Source Register 1 — FINTSRC1	794
481 FIQ Interrupt Source Register 2 — FINTSRC2	796
482 FIQ Interrupt Source Register 3 — FINTSRC3	797
483 Interrupt Priority Register 0 — IPR0	799
484 Interrupt Priority Register 1 — IPR1	800
485 Interrupt Priority Register 2 — IPR2	801
486 Interrupt Priority Register 3 — IPR3	802



487 Interrupt Priority Register 4 — IPR4	803
488 Interrupt Priority Register 5 — IPR5	804
489 Interrupt Priority Register 6 — IPR6	805
490 Interrupt Priority Register 7 — IPR7	806
491 Timer Performance Ranges	807
492 Timer Mode Register Control Bit Summary	808
493 Timer Responses to Register Bit Settings	810
494 Timer Registers	813
495 Timer Power Up Mode Settings	813
496 Timer Mode Register — TMRx	814
497 Timer Input Clock (TCLOCK) Frequency Selection	816
498 Timer Count Register — TCRx	817
499 Timer Reload Register — TRRx	817
500 Timer Interrupt Status Register — TISR	818
501 Watch Dog Timer Control Register — WDTCR	819
502 Watch Dog Timer Setup Register — WDTSR	819
503 Uncommon TMRx Control Bit Settings	820
504 Processor Send / Receive Queue Mapping	824
505 IMU Interrupt Summary	828
506 Fixed Priority Scheme for Door Bell Status bits and Circular Queue Attention Bits	829
507 Inter-processor Messaging Unit Registers	830
508 Door Bell Control Register — DBCR	832
509 Door Bell Enable Register — DBER	834
510 Door Bell Assertion Register — DBAR	835
511 Door Bell Enable Other Processor Register — DBEOR	835
512 Send Queue Put/Get Pointer Register 0 — SQPG0	836
513 Send Queue Control Register 0 — SQCR0	837
514 Send Queue Lower Base Address Register 0 — SQLBAR0	838
515 Send Queue Upper Base Address Register 0 — SQUBAR0	838
516 Receive Queue Put/Get Pointer Register 0 — RQPG0	839
517 Receive Queue Put/Get Pointer Register 0 — RQPG0	840
518 Receive Queue Lower Base Address Register 0 — RQLBAR0	840
519 Receive Queue Upper Base Address Register 0 — RQUBAR0	841
520 Send Queue Put/Get Pointer Register 1 — SQPG1	841
521 Send Queue Control Register 1 — SQCR1	842
522 Send Queue Lower Base Address Register 1 — SQLBAR1	843
523 Send Queue Upper Base Address Register 1 — SQUBAR1	843
524 Receive Queue Put/Get Pointer Register 1 — RQPG1	844
525 Receive Queue Put/Get Pointer Register 1 — RQPG1	845
526 Receive Queue Lower Base Address Register 1 — RQLBAR1	846
527 Receive Queue Upper Base Address Register 1 — RQUBAR1	846
528 Send Queue Put/Get Pointer Register 2 — SQPG2	847
529 Send Queue Control Register 2 — SQCR2	848
530 Send Queue Lower Base Address Register 2 — SQLBAR2	849
531 Send Queue Upper Base Address Register 2 — SQUBAR2	849
532 Receive Queue Put/Get Pointer Register 2 — RQPG2	850
533 Receive Queue Put/Get Pointer Register 2 — RQPG2	851
534 Receive Queue Lower Base Address Register 2 — RQLBAR2	852
535 Receive Queue Upper Base Address Register 2 — RQUBAR2	852
536 Send Queue Put/Get Pointer Register 3 — SQPG3	853
537 Send Queue Control Register 3 — SQCR3	854
538 Send Queue Lower Base Address Register 3 — SQLBAR3	854
539 Send Queue Upper Base Address Register 3 — SQUBAR3	855
540 Receive Queue Put/Get Pointer Register 3 — RQPG3	855
541 Receive Queue Put/Get Pointer Register 3 — RQPG3	856



542	Receive Queue Lower Base Address Register 3 — RQLBAR3	856
543	Receive Queue Upper Base Address Register 3 — RQUBAR3	857
544	IMU Test and Set Registers — IMUTSR[0:511]	860
545	SMBus Interface Pins	861
546	SMBus Command Encoding	863
547	SMBus Interface Registers for Configuration Space Access	867
548	SMBus Interface Registers for Memory Space Access	867
549	SMBus Status Byte Encoding	869
550	SMBus Register Summary	875
551	SMBus Controller Command Register — SM_CMD	875
552	SMBus Controller Byte Count Register — SM_BC	876
553	SMBus Controller ADDR3 Register — SM_ADDR3	876
554	SMBus Controller ADDR2 Register — SM_ADDR2	876
555	SMBus Controller ADDR1 Register Number — SM_ADDR1	877
556	SMBus Controller ADDR0 Register Number — SM_ADDR0	877
557	SMBus Controller Data Register — SM_DATA	878
558	SMBus Controller Status Register — SM_STS	878
559	UART Signal Descriptions	881
560	Divisor Values for Typical Baud Rates	886
561	UART Register Addresses as Offsets of a Base	888
562	UART Unit Registers	888
563	UART Register MMR Addresses	889
564	UART x Receive Buffer Register - (UxRBR)	890
565	UART x Transmit Holding Register - (UxTHR)	890
566	UART x Interrupt Enable Register - (UxIER)	891
567	UART x Interrupt Identification Register - (UxIIR)	892
568	Interrupt Identification Register Decode	893
569	UART x FIFO Control Register - (UxFCR)	894
570	UART x Line Control Register - (UxLCR)	896
571	UART x Modem Control Register - (UxMCR)	898
572	UART x Line Status Register - (UxLSR)	900
573	UART x Modem Status Register	903
574	UART x Modem Status Register - (UxMSR)	903
575	UART x Scratchpad Register - (UxSCR)	904
576	UART x Divisor Latch Low Register - (UxDLL)	905
577	UART x Divisor Latch High Register - (UxDLH)	905
578	UART x FIFO Occupancy Register - (UxFOR)	906
579	UART x Auto-Baud Control Register - (UxABR)	907
580	UART x Auto-Baud Count Register - (UxACR)	908
581	I ² C Bus Definitions	910
582	Modes of Operation	914
583	START and STOP Bit Definitions	915
584	Master Transactions	922
585	Slave Transactions	925
586	General Call Address Second Byte Definitions	927
587	I ² C Register Summary	932
588	I ² C Control Register x — ICRx	933
589	I ² C Status Register x — ISRx	935
590	I ² C Slave Address Register x — ISARx	937
591	I ² C Data Buffer Register x — IDBRx	938
592	I ² C Bus Monitor Register x — IBMRx	939
593	I ² C Manual Bus Control Register x — IMBCRx	940
594	General Purpose I/O Registers Addresses	942
595	GPIO Output Enable Register — GPOE	943
596	GPIO Input Data Register — GPID	944



597 GPIO Output Data Register — GPOD	946
598 Simple Time Based Counting of Events Example	951
599 Hardware Event Based Event Counting Example	953
600 Hardware Event Based Event Counting Example	955
601 Queue Depth Histogram Example	956
602 Head of Queue Histogram Example	957
603 PMON Internal Bus Memory Mapped Register Range Offsets	963
604 PMON Register Summaries	963
605 PMON Feature Enable Register - PMONEN	964
606 PMON Status Register - PMONSTAT	964
607 PMON Internal Bus Memory Mapped Register Range Offsets	965
608 PMON Register Summaries	966
609 PMON Command Register 0-7 - PMON_CMD [0:7]	967
610 PMON Event Register 0-7 - PMON_EVR [0:7]	971
611 PMON Status Register 0-7 - PMON_STS [0:7]	972
612 PMON DATA Register 7-0 - PMON_DATA [7:0]	974
613 Event Selection Code Summary	975
614 Intel® 81341 and 81342 I/O Processors PMON Clock Events	975
615 81341 and 81342 PMON Clock Events	976
616 81341 and 81342 PMON Threshold Events	976
617 DDR SDRAM Memory Controller Source Select Summary	977
618 DDR SDRAM Memory Controller Events	978
619 PCI Interface Events	981
620 PCI Express* Interface Summary	982
621 North Internal Bus Source Select Summary	983
622 North Internal Bus Initiator Events	983
623 South Internal Bus Source Select Summary	984
624 South Internal Bus Initiator Events	984
625 PCI Bus Frequency Initialization	987
626 CR_FREQ [1:0] Encoding	987
627 HS_FREQ Encoding	988
628 PCI-X Initialization Pattern ¹	988
629 Secondary Clock Output Control	989
630 Clock Pin Summary	991
631 Internal Bus Reset Summary	998
632 Reset Pin Summary	1000
633 Reset Strap Signals	1003
634 TLU TAP Controller Instruction Set	1014
635 IOP Device ID Register Field Definitions	1015
636 IOP Device ID Register Settings	1015
637 PMMR Base Address Register (PMMRBAR) Default Value	1022
638 Local Addresses for Integrated Peripherals	1022
639 ADMAx Base Address Offset	1025
640 Application DMA Unit	1025
641 IMU Base Address Offset	1027
642 Inter-Processor Messaging Unit	1027
643 SMCU Base Address Offset	1029
644 SRAM Memory Controller	1029
645 PBI Base Address Offset	1030
646 Peripheral Bus Interface Unit	1030
647 SC Base Address Offset	1030
648 System Controller Unit	1030
649 Internal Bus Bridge Base Address Offset	1031
650 Internal Bus Bridge	1031
651 DMCU Base Address Offset	1032



652 DDR SDRAM Memory Controller	1032
653 I/O Pad Control Base Address Offset.....	1033
654 I/O Pad Control Unit.....	1034
655 UART 0-1 Offset.....	1036
656 UART.....	1036
657 GPIO Offset.....	1037
658 GPIO	1037
659 I ² C 0-2 Offset.....	1037
660 I2C Unit.....	1037
661 Messaging Unit Offset.....	1038
662 Messaging Unit	1038
663 PMON Unit Base Address Offset.....	1040
664 PMON Unit	1040
665 PCI Function MMR Locations	1041
666 81341 and 81342 ATUX Configuration Space Base Address Offset.....	1042
667 Address Translation Unit Registers — ATUX	1043
668 81341 and 81342 ATUE Configuration Space Base Address Offset.....	1046
669 Address Translation Unit Registers — ATUE	1047
670 Intel® 81348 I/O Processor PCI Function Visibility	1051
671 Coprocessor Registers Assigned to Integrated Peripherals.....	1052
672 Coprocessor Register Locations	1053



Revision History

Date	Revision	Description
December 2007	002	Revised Chapter 7.0 , “DDR SDRAM Memory Controller” for 4GB reference.
September 2006	001	Launch Release.



1.0 Introduction

1.1 About This Document

This document is the authoritative and definitive reference for the external architecture of the Intel® 81341 and 81342 I/O Processors (81341 and 81342), with Intel XScale® microarchitecture¹.

Intel Corporation assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice. In particular, descriptions of features, timings, packaging, and pin-outs does not imply a commitment to implement them. In fact, this specification does not imply a commitment by Intel to design, manufacture, or sell the product described herein.

1.1.1 How To Read This Document

This document describes the product-specific features of the 81341 and 81342. Each chapter describes a different feature and starts with an overview followed by the theory of operation.

The reader should have a working understanding of the Peripheral Component Interconnect (PCI) Local Bus Specification, the PCI-X Addendum to the PCI Local Bus Specification and the PCI Express Specification. For more information, refer to the *PCI Local Bus Specification*, Revision 2.3, the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 2.0a, and the *PCI Express Specification*, Revision 1.0a.

1.1.2 Other Relevant Documents

1. Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual (Order Number: 273411), Intel Corporation
2. *PCI Local Bus Specification*, Revision 2.3 - PCI Special Interest Group
3. *PCI-X Addendum to the PCI Local Bus Specification*, Revision 2.0a - PCI Special Interest Group
4. *PCI Bus Power Management Interface Specification*, Revision 1.1 - PCI Special Interest Group
5. *PCI Express Specification*, Revision 1.0a - PCI Special Interest Group

1. ARM architecture compliant.



1.2 About the Single-Core Intel® 81341 I/O Processor

The 81341 1-core is a single-PCI device that integrates an Intel XScale® processor with intelligent peripherals including a PCI bus application bridge. The 81341 also supports two internal busses: north internal bus and south internal bus. With the two internal busses, transactions can take place simultaneously on each bus. The north internal bus provides the Intel XScale® processor with low latency access to either the DDR SDRAM Memory Controller, and the on-chip SRAM Memory Controller. Peripherals that generate large burst transactions are located on the south internal bus, thus allowing the Intel XScale® processor exclusive access to the north internal bus.

The 81341 1-core consolidates, into a single system:

- An Intel XScale® processor running at speed up to 1.5 GHz
- PCI - Local Memory Bus Address Translation Unit - PCI function 0
- Messaging Unit, PCI function 0
- Application Direct Memory Access (DMA) Controllers
- Peripheral Bus Interface Unit
- Integrated DDR2 Memory Controller
- Integrated SRAM Memory Controller
- Performance Monitor Unit (**PMON**)
- Two Programmable Timers on the Intel XScale® processor co-processor bus
- Watchdog Timer on the Intel XScale® processor co-processor bus
- Three I²C Bus Interface Units
- Two Serial Port Units
- Eight General Purpose Input Output (GPIO) ports
- Internal North Bus-South Bus Bridge

It is an integrated processor that addresses the needs of intelligent I/O Storage applications and helps reduce intelligent I/O system costs.

The 81341 support both a PCI-X 2.0 and a PCI Express interface. The PCI Bus is an industry standard, high performance, low latency system bus. The PCI Bus of the 81341 is capable of 133 MHz operation in PCI-X 2.0 mode as defined by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a. Also, the processor supports a 66 MHz conventional PCI mode as defined by the *PCI Local Bus Specification*, Revision 2.2. The addition of the Intel XScale® processors bring intelligence to the PCI bus application bridge. The 81341 supports a x8 PCI Express interface as defined in the *PCI Express Specification*, Revision 1.0.

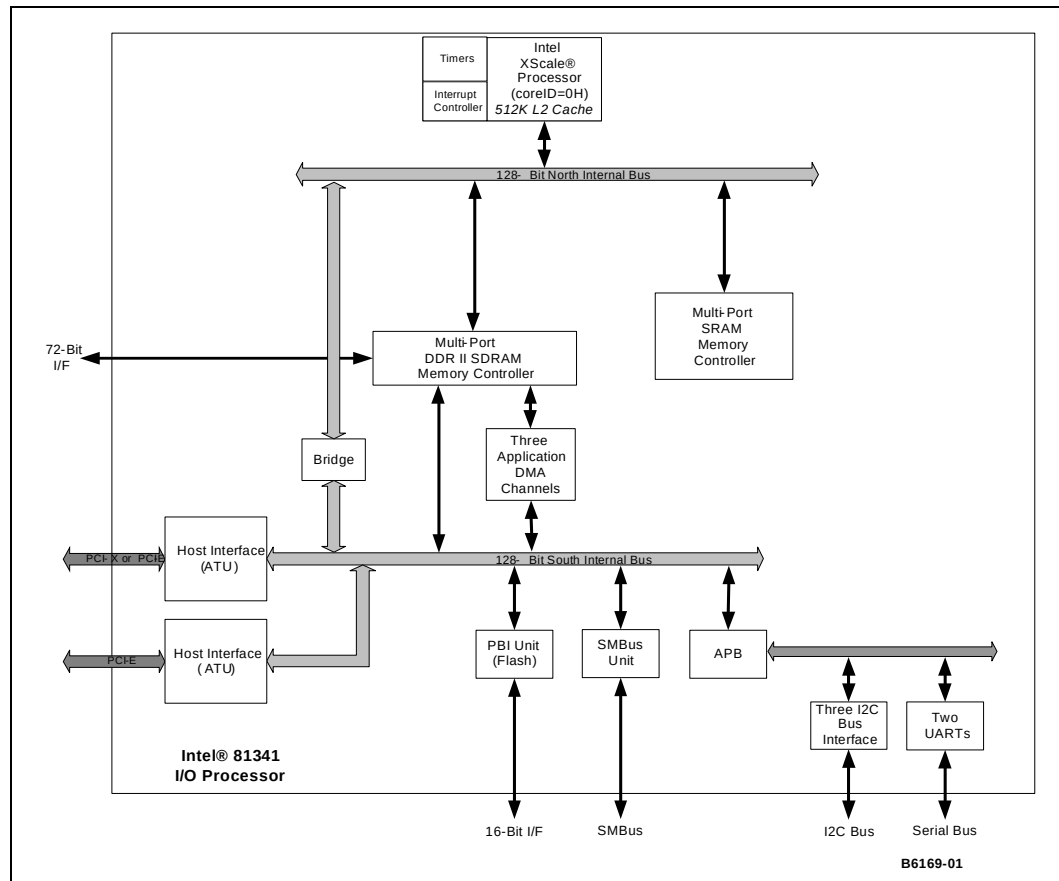
The 81341 is a single-function PCI device at reset. The host programming interface is presented as the Address Translation Unit (ATU) and the Messaging Unit (MU). The MU provides the messaging interface between the host processor and the 81341.

Both the address and data busses on the 81341 south internal bus are byte-wise parity protected.



Figure 1 is a block diagram of the 81341 1-core.

Figure 1. Single-Core Intel® 81341 I/O Processor Functional Block Diagram





1.3 Single-Core Intel® 81341 I/O Processor Features

The 81341 1-core combines two Intel XScale® processors with powerful new features to create an intelligent I/O storage processor. This single- or multi-function PCI device is fully compliant with the *PCI-X 2.0* and *PCI Express 1.0* specifications. The 81341-specific features include:

- Address Translation Unit
- Messaging Unit
- Peripheral Bus Interface Unit (PBI)
- **Three I²C Bus Interface Unit**
- SRAM Memory Controller
- Application DMA Controllers
- DDR2 Memory Controller
- Internal Bus Bridge
- **Performance Monitoring Unit (PMON)**
- **Two** Serial Port Units (UARTs)
- Address and Data Bus Parity Protection
- Programmable Timers
- Watchdog Timers

The 81341 core is based upon an Intel XScale® processors. The core processor operates at a maximum frequency of 1.5 GHz. The instruction cache is 32 Kbytes in size and is 32-way set associative. Also, the core processor includes a data cache that is 32 Kbytes and is 32-way set associative and a mini data cache that is 2 Kbytes and is 2-way set associative. The Intel XScale® processor supports a unified 512-KByte Level 2 (L2) cache and is 8-way set associative.

The 81341 includes sixteen General Purpose I/O (GPIO) pins.



1.4 About the Dual-Core Intel® 81342 I/O Processor

The 81342 2-core is a single-PCI device that integrates two Intel XScale® processors with intelligent peripherals including a PCI bus application bridge. 81342 2-core also supports two internal busses: North internal bus and South internal bus. With the two internal busses, transactions can take place simultaneously on each bus. The north internal bus provides the two Intel XScale® processors with low latency access to either the DDR SDRAM Memory Controller, and the on-chip SRAM Memory Controller. Peripherals that generate large burst transactions are located on the south internal bus, thus allowing the two Intel XScale® processors exclusive access to the north internal bus.

The 81342 2-core consolidates, into a single system:

- Two Intel XScale® processors running at speed up to 1.5 GHz
- PCI - Local Memory Bus Address Translation Unit
- Messaging Unit, function 0 programming interface
- Inter-Processor Communication
- Inter-Processor Messaging Unit
- Application Direct Memory Access (DMA) Controllers
- Peripheral Bus Interface Unit
- Integrated DDR2 Memory Controller
- Integrated SRAM Memory Controller
- Performance Monitor (**PMON**)
- **Two** Programmable Timers on the Intel XScale® processor co-processor bus
- Watchdog Timer on the Intel XScale® processor co-processor bus
- **Three** I²C Bus Interface Units
- **Two** Serial Port Units
- **Eight** General Purpose Input Output (GPIO) ports
- Internal North Bus-South Bus Bridge

It is an integrated processor that addresses the needs of intelligent I/O Storage applications and helps reduce intelligent I/O system costs.

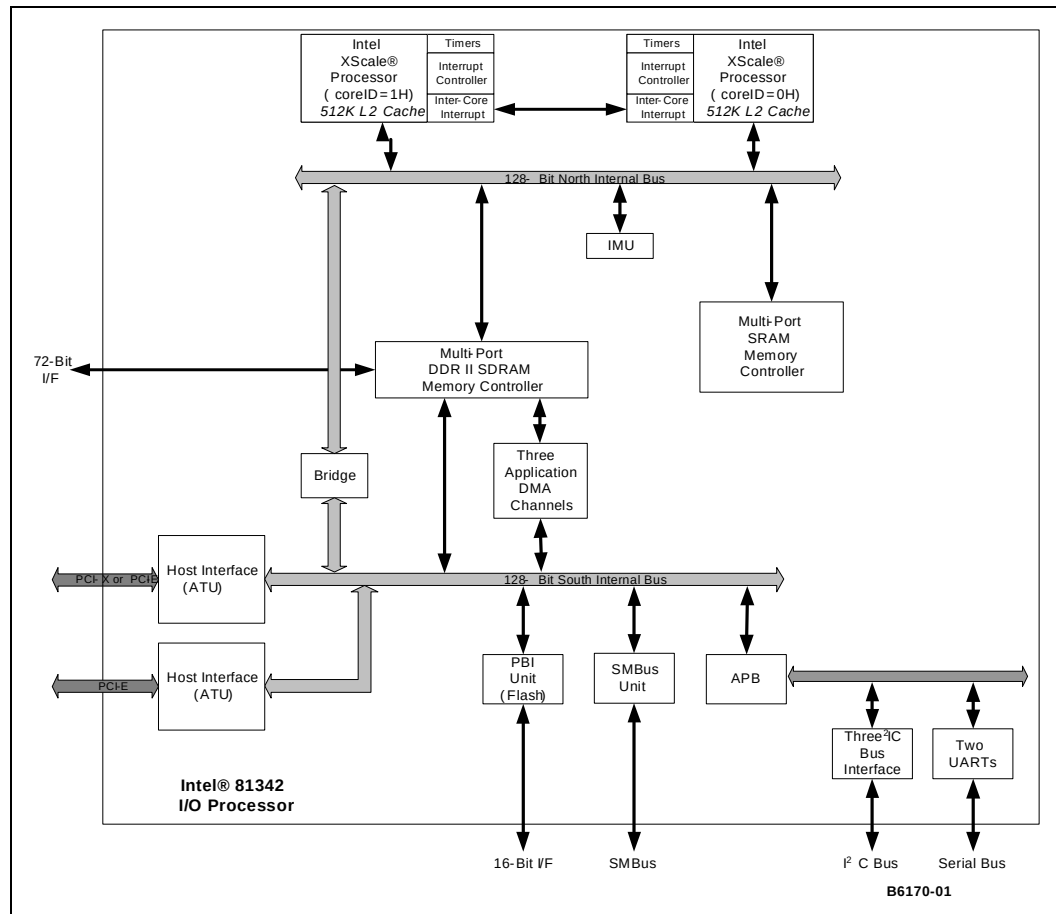
The 81342 2-core support both a PCI-X 2.0 and a PCI Express interface. The PCI Bus is an industry standard, high performance, low latency system bus. The 81342 2-core's PCI Bus is capable of 133 MHz operation in PCI-X 2.0 mode as defined by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a. Also, the processor supports a 66 MHz conventional PCI mode as defined by the *PCI Local Bus Specification*, Revision 2.2. The addition of the Intel XScale® processors bring intelligence to the PCI bus application bridge. The 81342 2-core supports a x8 PCI Express interface as defined by the *PCI Express Specification*, Revision 1.0.

The 81342 2-core is a single-function PCI device at reset. The host programming interface is presented as the Address Translation Unit (ATU) and the Messaging Unit (MU). The MU provides the messaging interface between the host processor and the 81342 2-core.

Both the address and data busses on the 81342 2-core south internal bus are byte-wise parity protected.

Figure 2 is a block diagram of the 81342 2-core.

Figure 2. Dual-Core Intel® 81342 I/O Processor Functional Block Diagram





1.5 Dual-Core Intel® 81342 I/O Processor Features

The 81342 2-core combines two Intel XScale® processors with powerful new features to create an intelligent I/O storage processor. This single- or multi-function PCI device is fully compliant with the *PCI-X 2.0 and PCI Express 1.0 specifications*. 81342 2-core-specific features include:

- Address Translation Unit
- Messaging Unit
- Peripheral Bus Interface Unit (PBI)
- **Three I²C Bus Interface Units**
- SRAM Memory Controller
- Application DMA Controllers
- Internal Bus Bridge
- DDR2 Memory Controller
- **Performance Monitoring Unit (PMON)**
- **Two** Serial Port Units (UARTs)
- Address and Data Bus Parity Protection
- Inter-Processor Communication
- Programmable Timers
- Watchdog Timer per core
- Inter-Processor Messaging Unit

The 81342 2-core core is based upon two Intel XScale® processors. The core processor operates at a maximum frequency of 1.5 GHz. The instruction cache is 32 Kbytes in size and is 32-way set associative. Also, the core processor includes a data cache that is 32 Kbytes and is 32-way set associative and a mini data cache that is 2 Kbytes and is 2-way set associative. Both Intel XScale® processors support unified 512-KByte Level 2 (L2) cache and is 8-way set associative.

The 81342 2-core includes sixteen General Purpose I/O (GPIO) pins.

The subsections that follow briefly overview each feature. Refer to the appropriate chapter for full technical descriptions.

1.5.1 Host Interface

The The 81341 1-core provides both a PCI-X 2.0 and a PCI Express interface. Either interface can be connected upstream to the Host processor. Either interface on 81341 and 81342 is setup as a single-function PCI device. The upstream PCI interface is selected using an external strap. The Address Translation Unit (ATU) and the Messaging Unit (MU) provide the programming interface between the host processor and the 81341 and 81342.

The 81342 2-core provides both a PCI-X 2.0 and a PCI Express interface. Either interface can be connected upstream to the Host processor. Either interface on 81342 2-core is setup as a single-function PCI device. The upstream PCI interface is selected using an external strap. The Address Translation Unit (ATU) and the Messaging Unit (MU) provide the programming interface between the host processor and the 81342 2-core.

1.5.2 Intel XScale® Processor

The Intel XScale® processor operates at a maximum frequency of 1.5 GHz. The instruction cache is 32 Kbytes in size and is 32-way set associative. Also, the processor includes a data cache that is 32 Kbytes and is 32-way set associative. The Intel XScale® processor supports a unified 512-KByte Level 2 (L2) cache and is 8-way set associative.

1.5.3 Internal Busses

The 81341 and 81342 is architected around two internal busses: north internal bus and south internal bus. The two busses use the same bus protocol. The north internal bus is 128-bit wide and operates at speed up to 400 MHz. The north internal bus connects the two Intel XScale® processors, which have direct access to the DDR SDRAM. The north internal bus is designed to provide the two Intel XScale® processors with low latency access.

The south internal bus is 128-bits wide and operates at speeds up to 400 MHz. The south internal bus provides data paths for large DMA generated burst transactions.

Both the internal address and data busses on the south internal bus are parity protected on a byte-wise basis. The point-to-point interfaces between the agents connected to the DDR SDRAM Memory Controllers are also parity protected on a byte-wise basis.

1.5.4 Application DMA Controller

There are three Application DMA Channels. The Application DMA controller is dual-ported - with one of its ports connected to the south internal bus and the other port to the DDR SDRAM Memory controller. This Application DMA Controller allows low-latency, high-throughput data transfers between PCI bus agents and the DDR memory. The DMA controller also allows data transfer between DDR Memory. The DMA Controller supports chaining and unaligned data transfers. It is programmable through the Intel XScale® processor and the host processor.

Each Application DMA channel can also be programmed to operate as an XOR Engine that provides low-latency, high-throughput data transfer capability between the ADMA unit, the local DDR memory and the Primary PCI bus. It executes data transfers from and to the local DDR memory, from the Primary PCI bus to the local DDR memory, or from the local DDR memory to the Primary PCI bus. The XOR unit performs XOR operations, computes parity, generates and verifies an eight byte Data Integrity field which includes a 16- /32- bit Data Guard, performs memory block fills, and provides the necessary programming interface.

The Intel XScale® processor supports L2 cache hardware coherency. Software can take advantage of the hardware coherency support by placing the ADMA descriptors in coherent memory space. The ADMA provides a control bit to enable this feature, and software must also set up the coherent memory regions by programming the appropriate page table entries. Refer to the "Memory Management" Chapter of the Intel XScale® processor Specification for more details.

1.5.5 Address Translation Unit

The Address Translation Unit (ATU) allows PCI transactions direct access to the local DDR SDRAM memory. The ATU provides the interface for the RAID Controller PCI function. The ATU supports transactions between PCI address space and the internal address space. Address translation is controlled through programmable registers accessible from both the PCI interface and the Intel XScale® processor. Dual access to registers allows flexibility in mapping the two address spaces. The ATU also supports the extended capability configuration headers.



1.5.6 Messaging Unit

The Messaging Unit (MU) provides data transfer between the PCI system and the 81341 and 81342. It uses interrupts to notify each system when new data arrives. The MU has four messaging mechanisms: Message Registers, Doorbell Registers, Circular Queues, and Index Registers. Each allows a host processor or external PCI device and the 81341 and 81342 to communicate through message passing and interrupt generation. The MU in conjunction with the ATU exists as the PCI Interface for PCI function 0 when function 0 is setup as a RAID Controller.

1.5.7 DDR Memory Controller

The DDR Memory Controller allows direct control of 400/533 MHz DDR2 SDRAM memory subsystem. It features programmable chip selects and support for error correction codes (ECC). The DDR Memory Controller is multi-ported with the following interfaces: south internal bus, DMA controller, north internal bus. The memory controller interface configuration support includes Unbuffered DIMMs, Registered DIMMs, and discrete DDR SDRAM devices.

1.5.8 SRAM Memory Controller

The SRAM Memory Controller allows direct control of a 1 MByte single-ported SRAM memory subsystem. It supports error correction codes (ECC). The SRAM can be used to store firmware code or for general purpose data storage.

1.5.9 Peripheral Bus Interface

The Peripheral Bus Interface Unit is a data communication path to the Flash memory components or other peripherals of 81341 and 81342 hardware system. The PBI includes support for either 8/16 bit devices. To perform these tasks at high bandwidth, the bus features a burst transfer capability which allows successive 8/16-bit data transfers.

1.5.10 Performance Monitoring Unit

The Performance Monitoring Unit allows various events on the 81341 and 81342 to be monitored.



1.5.11 I²C Bus Interface Unit

There are three I²C (Inter-Integrated Circuit) Bus Interface Units that allow the Intel XScale® processor to serve as a master and slave device residing on the I²C bus. The I²C unit uses a serial bus developed by Philips Semiconductor consisting of a two-pin interface. The bus allows 81341 and 81342 to interface to other I²C peripherals and microcontrollers for system management functions. It requires a minimum of hardware for an economical system to relay status and reliability information on the I/O subsystem to an external device. Also refer to *I²C Peripherals for Microcontrollers* (Philips Semiconductor).

1.5.12 UART Unit

The 81341 and 81342 includes two UART units. The UART Unit allows the two Intel XScale® processors to serve as a master and slave device residing on the UART bus. The UART unit uses a serial bus consisting of a two-pin interface. The bus allows 81341 and 81342 to interface to other peripherals and microcontrollers. Also refer to *16550 Device spec* (National Semiconductor).

1.5.13 Interrupt Controller Unit

Each Intel XScale® processor supports an Interrupt Controller Unit. The Interrupt Controller Unit (ICU) aggregates interrupt sources both external and internal of sources of 81341 and 81342 to the Intel XScale® processor processor. The ICU supports high performance interrupt processing with direct interrupt service routine vector generation on a per source basis. Each source has programmability for masking, core processor interrupt input, and priority.

1.5.14 Internal Bus System Controller

Each internal bus (north and south) employs a internal System Controller. The internal System Controller observes all the address or data bus request from requestors and completors connected to the internal bus. The internal System Controller includes features to handle: internal address bus arbitration, internal data bus arbitration, framing Address bus cycles, framing Data bus cycles, and provides the shared address and shared data paths from/to units.

1.5.15 Inter-Processor Communication

Each Intel XScale® processor can interrupt or issue a reset to the second Intel XScale® processor. Each core can generate up to thirty two interrupts to the second core.

1.5.16 Inter-Processor Messaging Unit

The Inter-Processor Messaging unit provides a method for the two integrated Intel XScale® processor to efficiently communicate. The hardware provides two communication mechanisms: Doorbell Registers and Circular Queues.

1.5.17 Timers

The 81341 and 81342 supports two programmable 32-bit timers per processor. The 81341 and 81342 also supports one watchdog timer per processor.

1.5.18 GPIO

The 81341 and 81342 includes sixteen General Purpose I/O (GPIO) pins.



1.6 Terminology and Conventions

1.6.1 Representing Numbers

All numbers in this document can be assumed to be Base10 unless designated otherwise. In text, numbers in Base16 are represented as "nnnH", where the "H" signifies hexadecimal. In pseudo code descriptions, hexadecimal numbers are represented in the form 0x1234ABCD. Binary numbers are not explicitly identified but are assumed when bit operations or bit ranges are used.

1.6.2 Fields

A *reserved* field is a field that may be used by an implementation. When the initial value of a reserved field is supplied by software, this value must be zero. Software should not modify reserved fields or depend on any values in reserved fields.

A *read/write* field can be written to a new value following initialization. This field can always be read to return the current value.

A *read only* field can be read to return the current value. Writes to *read only* fields are treated as no-op operations and does not change the current value nor result in an error condition.

A *read/clear* field can also be read to return the current value. A write to a *read/clear* field with the data value of 0 causes no change to the field. A write to a *read/clear* field with a data value of 1 causes the field to be cleared (reset to the value of 0). For example, when a *read/clear* field has a value of F0H, and a data value of 55H is written, the resultant field is A0H.

A *read/set* field can also be read to return the current value. A write to a *read/set* field with the data value of 0 causes no change to the field. A write to a *read/set* field with a data value of 1 causes the field to be set (set to the value of 1). For example, when a *read/set* field has a value of F0H, and a data value of 55H is written, the resultant field is F5H.

A *writeonce/readonly* field can be written to a new value **once** following initialization. After this write has occurred, the *writeonce/readonly* field treats all subsequent writes as no-op operations and does not change the current value or result in an error condition. The field can always be read to return the current value.



1.6.3 Specifying Bit and Signal Values

The terms *set* and *clear* in this specification refer to bit values in register and data structures. When a bit is set, its value is 1; when the bit is clear, its value is 0. Likewise, *setting* a bit means giving it a value of 1 and *clearing* a bit means giving it a value of 0.

The terms *assert* and *deassert* refer to the logically active or inactive value of a signal or bit, respectively.

1.6.4 Signal Name Conventions

All signal names use the PCI signal name convention of using the “#” symbol at the end of a signal name to indicate that the signal’s active state occurs when it is at a low voltage. The absence of the “#” symbol indicates that the signal’s active state occurs when it is at a high voltage.

1.6.5 Terminology

To aid the discussion of the 81341 and 81342 architecture, the following terminology is used:

Downstream	At or toward a PCI bus with a higher number (after configuration)
DWORD	32-bit data word
QWORD	64-bit data word
word	32-bit data word
Host processor	Processor located upstream from the 81341 and 81342
Local processor	Intel XScale® processor within the 81341 and 81342
Local bus	81341 and 81342 Internal Bus
Local memory	Memory subsystem on the Intel XScale® processor DDR SDRAM or Peripheral Bus Interface busses
Upstream	At or toward a PCI bus with a lower number (after configuration)



2.0 Address Translation Unit (PCI-X)

This chapter describes the operation modes, setup, and implementation of the module which interfaces between the PCI bus and the Intel® 81341 and 81342 I/O Processors (81341 and 81342) internal bus.

2.1 Overview

As indicated in [Figure 3](#), the Address Translation Unit (ATU) — the interface between the PCI bus and the on-chip internal bus — consists of the Address Translation Unit (ATU) and the Expansion ROM Unit.

The ATU supports both inbound and outbound address translation. The ATU provides access between the PCI bus and the 81341 and 81342 internal bus.

Transactions initiated on the PCI bus and targeted at the 81341 and 81342 internal bus are referred to as *inbound transactions* (PCI to internal bus). Transactions initiated on the 81341 and 81342 internal bus and targeted at the PCI bus are referred to as *outbound transactions* (internal bus to PCI). The ATU accepts multiple inbound or outbound transactions and processes them simultaneously.

During inbound transactions, the ATU converts PCI addresses (initiated by a PCI bus master) to Internal Bus Addresses and initiates the data transfer on the 81341 and 81342 internal bus. During outbound transactions, the ATU converts internal bus addresses to PCI addresses and initiates the data transfer on the PCI bus.

The Expansion ROM provides the PCI mechanism for downloading device/board driver code during system boot sequence. It consists of a separate inbound address range which accesses a Flash EPROM device connected through the 81341 and 81342 memory controller. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details of Expansion ROM usage.

The Address Translation Unit and the Expansion ROM Translation Unit represent a single function of the multi-function 81341 and 81342 PCI device.

The ATU supports the following PCI operating modes and bus widths delivering up to 2133 Mbytes/sec of bandwidth:

- Conventional Modes: PCI 33, PCI 66
- PCI-X Modes: Mode 1 (PCI-X 66, PCI-X 133), Mode 2 (PCI-X 266)
- Bus Widths: 64-bit, 32-bit

In Mode 2, all transaction phases are ECC protected (when enabled). In the remaining PCI-X and Conventional modes supported by 81341 and 81342, all transaction phases are parity protected (when enabled).

On the internal interface, the ATU implements the 81341 and 81342 internal bus protocol which provides for a maximum of 4800 Mbytes/sec of bandwidth.

The 81341 and 81342 meets the standard requirements to be considered "Hot-Swap Silicon" detailed in the *Compact PCI Hot-Swap Specification*, Revision 2.1.

Address and data are protected by byte-wise parity on the internal bus.

The ATU includes four extended capability headers that implement Power Management capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1, MSI capability as defined by *PCI Local Bus Specification*, Revision 2.3, Hot-Swap capability as defined by the *Compact PCI Hot-Swap Specification*, Revision 2.1, and PCI-X capability as defined by *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0.

The functionality of the ATU is described in the following sections. The ATU has a memory-mapped register interface that is visible from either the PCI interface, the internal bus interface, or both.

Figure 3. ATU Block Diagram

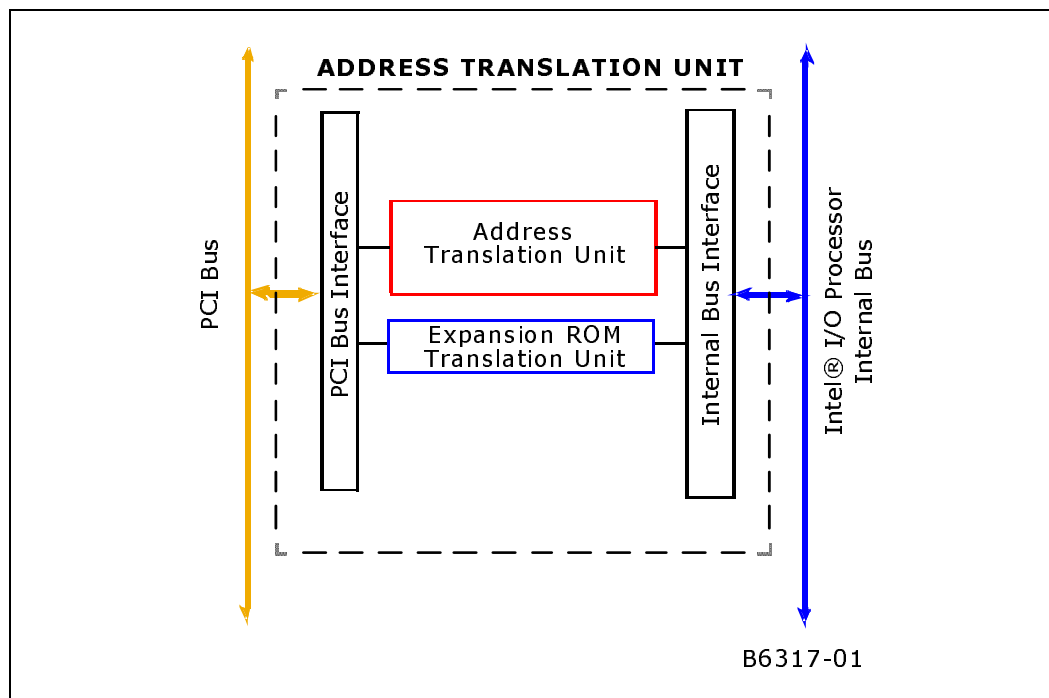
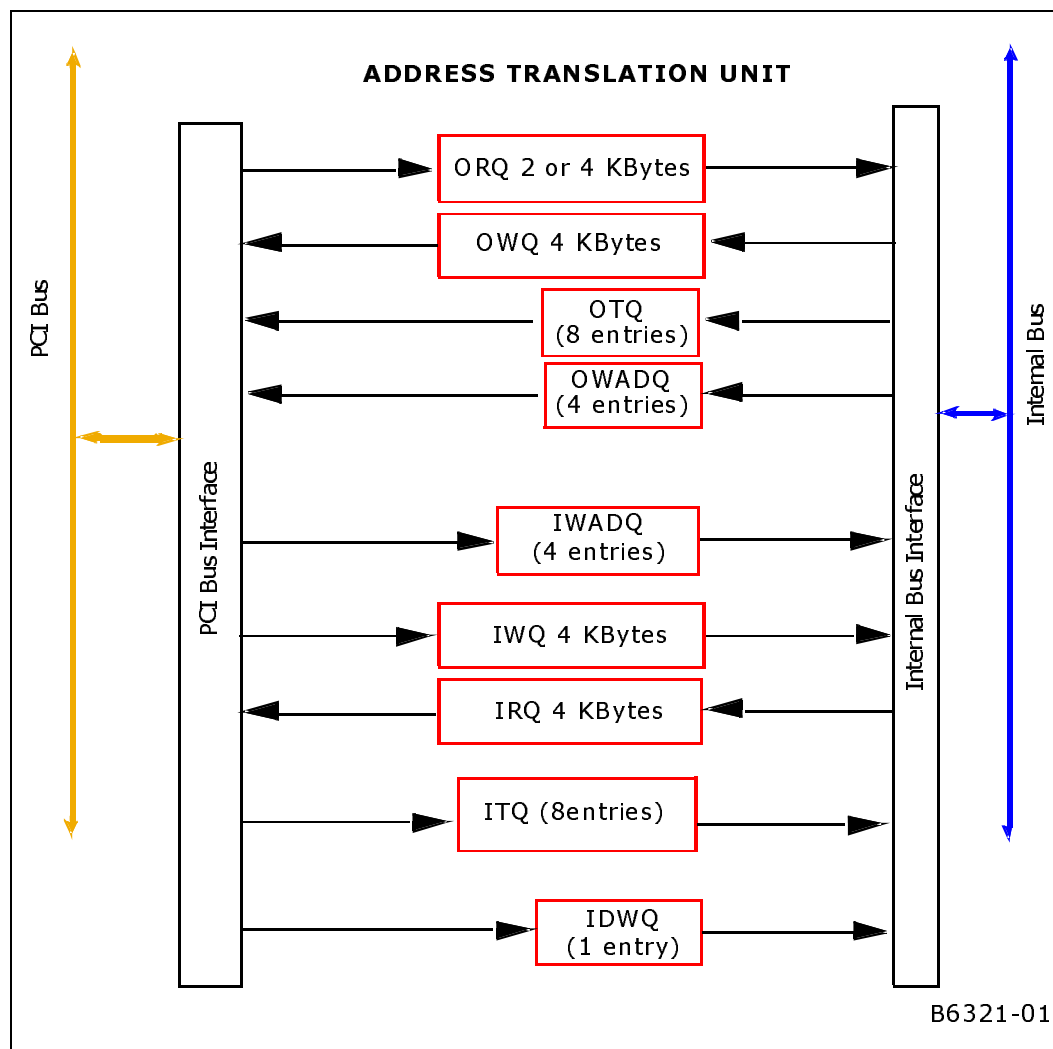


**Figure 4. ATU Queue Architecture Block Diagram**



2.2 ATU Address Translation

The ATU allows PCI masters on the PCI bus to initiate transactions to the 81341 and 81342 internal bus and allows the Intel XScale® processor (ARM* architecture compliant) to initiate transactions to the PCI bus.

The ATU implements an address windowing scheme to determine which addresses to claim and translate to the destination bus.

- The address windowing mechanism for inbound translation is described in [Section 2.2.1.1, “Inbound Address Translation” on page 61](#)
- The address windowing mechanism for outbound translation is described in [Section 2.2.2, “Outbound Transactions- Single Address Cycle \(SAC\) Internal Bus Transactions” on page 72](#) and [Section 2.2.3, “Outbound Write Transaction” on page 77](#)

The ATU has the ability to accept up to eight inbound PCI read transactions and four inbound PCI write transactions simultaneously. Also, the ATU has the ability to accept up to eight outbound internal bus read transactions and four outbound internal bus write transactions simultaneously. Refer to [Figure 4](#) and [Section 2.6](#) for details of the ATU queue architecture.

The ATU unit allows for recognition and generation of multiple PCI cycle types. [Table 1](#) shows the PCI and PCI-X commands supported for both inbound and outbound ATU transactions. The type of operation seen by the ATU on inbound transactions is determined by the PCI master who initiates the transaction. Claiming an inbound transaction depends on the address range programmed within the inbound translation window. The type of transaction used by the ATU on outbound transactions generated by the core processor is determined by the internal bus address and the outbound windowing scheme.

ATU supports the 64-bit addressing specified by the *PCI Local Bus Specification*, Revision 2.3. This 64-bit addressing extension is supported for both inbound and outbound data transactions. This is in addition to the 64-bit data extensions supported by the 81341 and 81342.

ATU does not support exclusive access using the PCI LOCK# signal. Also, the ATU does not insure atomicity for outbound transactions.

Note:

In conventional PCI Mode, the ATUX will pre-fetch data for read transactions on the internal bus as defined in [Section 2.6.1.2, “Inbound Read Queue Structure” on page 89](#). For internal bus targets that are not capable of supporting large byte-counts as indicated in [Table 8, “Inbound Read Prefetch Data Sizes” on page 89](#), these internal bus targets must be accessed using a non-prefetchable PCI window. This requirement also includes access made to the north internal bus targets via the internal bus bridge. The internal bus bridge supports 32-byte data queues, and any access made via the internal bus bridge that is greater than 32 bytes will result in an internal bus target abort.

**Table 1. ATU Command Support**

PCI Command Encoding	PCI Command Type	PCI-X Command Type	Claimed on Inbound Transactions on PCI Bus	Generated by Outbound Transactions on PCI Bus	Valid Internal Bus Command
0000	Interrupt Acknowledge	Interrupt Acknowledge	No	No	Reserved
0001	Special Cycle	Special Cycle	No	No	Reserved
0010	I/O Read	I/O Read	No	Yes	Reserved
0011	I/O Write	I/O Write	No	Yes	Reserved
0100	Reserved	Reserved	No	No	Reserved
0101	Reserved	Device ID Message ¹	No	No	Reserved
0110	Memory Read	Memory Read DWORD	Yes	Yes	Read
0111	Memory Write	Memory Write	Yes	Yes	Write
1000	Reserved	Alias to Memory Read Block	Yes	No	Read
1001	Reserved	Alias to Memory Write Block	Yes	No	Write
1010	Configuration Read	Configuration Read	Yes	Yes	Read
1011	Configuration Write	Configuration Write	Yes	Yes	Write
1100	Memory Read Multiple	Split Completion	Yes	Yes	Split Completion
1101	Dual Address Cycle	Dual Address Cycle	Yes	Yes	Reserved
1110	Memory Read Line	Memory Read Block	Yes	Yes²	Read
1111	Memory Write and Invalidate	Memory Write Block	Yes	Yes	Write

Notes:

1. PCI-X mode 2 only
2. PCI-X mode only

Inbound and outbound ATU transactions are best described by the data flows used on the PCI bus and the 81341 and 81342 internal bus during read and write operations. The following sections describe read and write operations for inbound ATU transactions (PCI to internal bus) and outbound transactions (internal bus to PCI).



2.2.1 Inbound Transactions

Inbound transactions which target the ATU are translated and executed on the 81341 and 81342 internal bus. As a PCI target, the ATU is capable of accepting all PCI memory read and write operations as either a 32-bit or a 64-bit PCI target. In the conventional PCI mode *Memory Write* and *Memory Write and Invalidate* operations are performed as posted operations and all memory read operations are performed as delayed reads. In the PCI-X mode *Memory Write*, *Memory Write Block*, and *Alias to Memory Write Block* operations are performed as posted operations and *Memory Read DWORD*, *Memory Read Block*, and *Alias to Memory Read Block* operations are executed as split transactions. The ATU is capable of accepting configuration read and write cycles. In the conventional PCI mode, *Configuration Writes* are performed as delayed memory write operations and *Configuration Reads* are performed as delayed read operations. In the PCI-X mode, both *Configuration Writes* and *Configuration Reads* are performed as split transactions.

Inbound memory write transactions have their addresses entered into the inbound write address queue (IWADQ) and data entered into the inbound write data queue (IWQ). The IWQ/IWADQ pair are capable of holding up to 4 write operations up to the size of the data queue. Inbound configuration writes use the inbound delayed write queue (IDWQ) for address and data. Refer to [Section 2.6](#) for details of queue operation. Inbound read operations (memory and configuration) have their address entered into the inbound transaction queue (ITQ) and the data is returned to the PCI master in the inbound read queue (IRQ). The ITQ is capable of holding up to 8 delayed read requests (split read requests when operating in the PCI-X mode).

In the conventional PCI mode, for inbound transactions, the ATU is a slave on the PCI bus and is a requester on the internal bus. PCI slave operation is defined in the *PCI Local Bus Specification*, Revision 2.3. In the PCI-X mode, for inbound transactions, the ATU initially is a target on the PCI bus and becomes an initiator when performing split completion transactions, and is an initiator on the internal bus.

Operation of the internal bus is defined in [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#). Specific operation of the ATU as master on the internal bus is defined in [Section 2.2.6](#).



2.2.1.1 Inbound Address Translation

The ATU allows external PCI bus initiators to directly access the internal bus. These PCI bus initiators can read or write 81341 and 81342 memory-mapped registers or 81341 and 81342 local memory space. The process of inbound address translation involves two steps:

1. Address Detection.
 - a. Determine when the 32-bit PCI address (64-bit PCI address during DACs) is within the address windows defined for the inbound ATU.
 - b. Claim the PCI transaction with medium DEVSEL# timing in the conventional PCI mode and with Decode A DEVSEL# timing in the PCI-X mode.
2. Address Translation.
 - a. Translate the 32-bit PCI address (lower 32-bit PCI address during DACs) to a 36-bit 81341 and 81342 internal bus address.

The ATU uses the following registers in inbound address window 0 translation:

- Inbound ATU Base Address Register 0
- Inbound ATU Limit Register 0
- Inbound ATU Translate Value Register 0
- Inbound ATU Upper Translate Value Register 0

The ATU uses the following registers in inbound address window 1 translation:

- Inbound ATU Base Address Register 1
- Inbound ATU Limit Register 1
- Inbound ATU Translate Value Register 1
- Inbound ATU Upper Translate Value Register 1

The ATU uses the following registers in inbound address window 2 translation:

- Inbound ATU Base Address Register 2
- Inbound ATU Limit Register 2
- Inbound ATU Translate Value Register 2
- Inbound ATU Upper Translate Value Register 2

The ATU uses the following registers in inbound address window 3 translation:

- Inbound ATU Base Address Register 3
- Inbound ATU Limit Register 3
- Inbound ATU Translate Value Register 3
- Inbound ATU Upper Translate Value Register 3

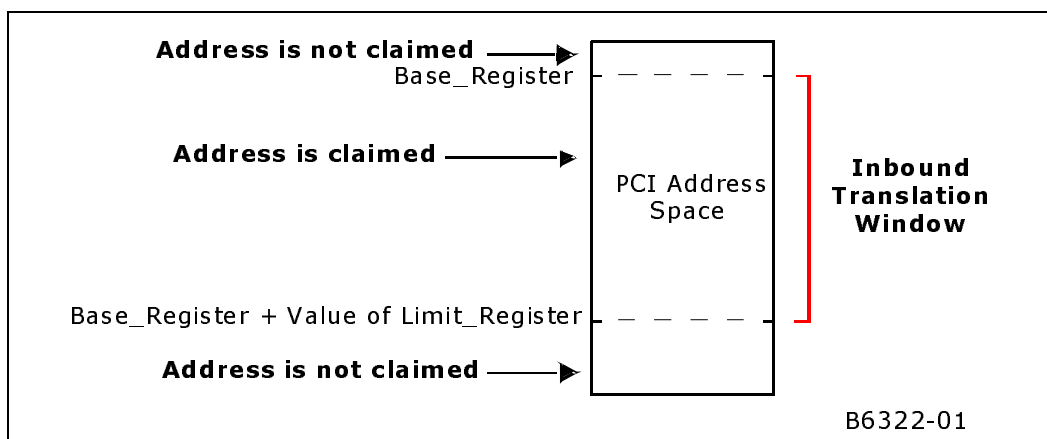
Inbound address detection is determined from the 32-bit PCI address, (64-bit PCI address during DACs) the base address register and the limit register. In the case of DACs none of the upper 32-bits of the address is masked during address comparison. The algorithm for detection is:

Equation 1. Inbound Address Detection

When $\text{PCI\_Address}[31:0] \& \text{Limit\_Register}[31:0] == \text{Base\_Register}[31:0]$ and $\text{PCI\_Address}[63:32] == \text{Base\_Register}[63:32]$ (for DACs only) the PCI Address is claimed by the Inbound ATU.

Figure 5 shows an example of inbound address detection.

Figure 5. Inbound Address Detection



The incoming 32-bit PCI address (lower 32-bits of the address in case of DACs) is bitwise ANDed with the associated inbound limit register. When the result matches the base register (and upper base address matches upper PCI address in case of DACs), the inbound PCI address is detected as being within the inbound translation window and is claimed by the ATU.

Note: By default, the first 8 Kbytes of the ATU inbound address translation window 0 are reserved for the Messaging Unit. See [Chapter 4.0, “Messaging Unit”](#).

Once the transaction is claimed, the address must be translated from a PCI address to a 36-bit internal bus address. In case of DACs upper 32-bits of the address is simply discarded and only the lower 32-bits are used during address translation. The algorithm is:

Equation 2. Inbound Translation

$$\text{Intel}^{\circledR} \text{ 81341 and 81342 I/O Processors Internal Bus Address} = ((\text{PCI\_Address}[31:0] \& \sim \text{Limit\_Register}[31:0]) \mid \text{ATU\_Translate\_Value\_Register}[31:0]) \mid (\text{ATU\_Upper\_Translate\_Value\_Register}[3:0] \ll 32).$$

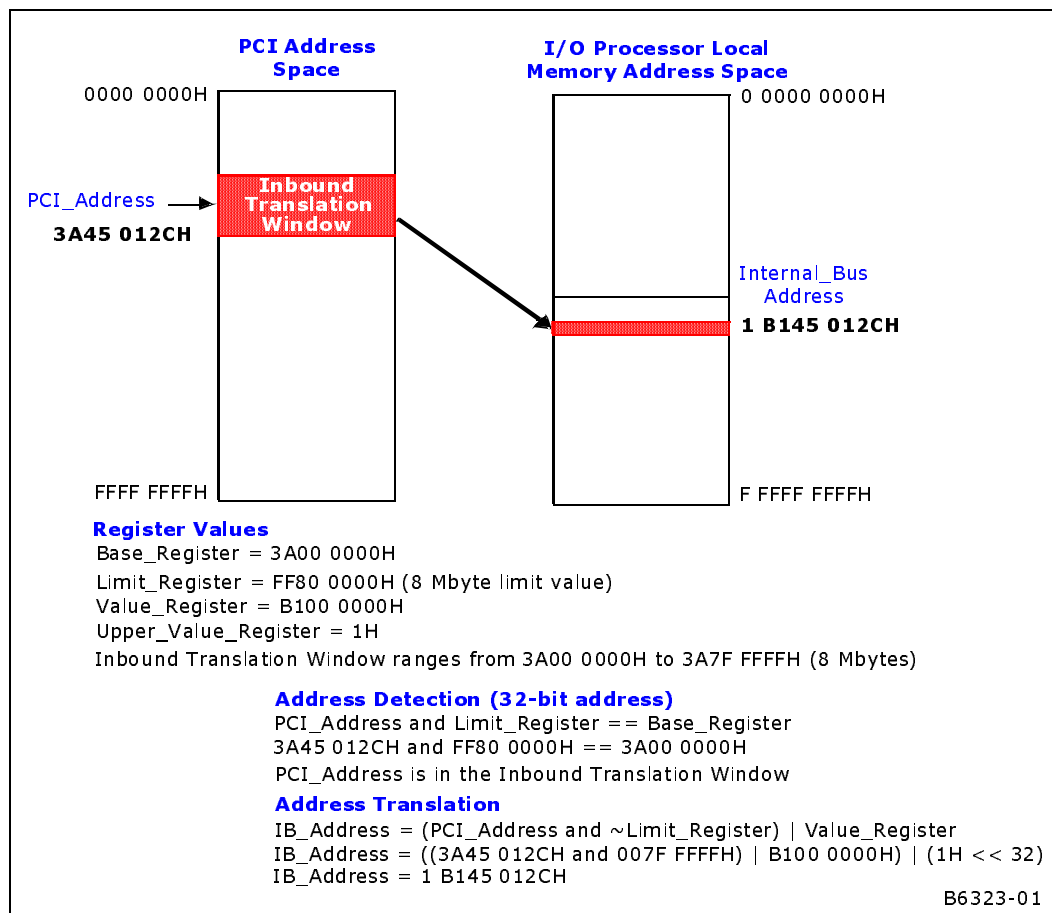
The incoming 32-bit PCI address (lower 32-bits in case of DACs) is first bitwise ANDed with the bitwise inverse of the limit register. This result is bitwise ORed with the ATU Translate Value, which is then ORed with the 4-bit ATU Upper Translate Value left shifted by 32; the result is the 36-bit internal bus address. This translation mechanism is used for all inbound memory read and write commands excluding inbound configuration read and writes. Inbound configuration cycle translation is described in [Section 2.2.1.4, “Inbound Configuration Cycle Translation”](#) on page 69.

In the PCI mode for inbound memory transactions, the only burst order supported is Linear Incrementing. For any other burst order, the ATU signals a Disconnect after the first data phase. The PCI-X supports linear incrementing only, and hence the above situation is never encountered in the PCI-X mode.



Figure 6 shows an inbound translation example for 32-bit addressing. This example would hold true for an inbound transaction from PCI bus.

Figure 6. Inbound Translation Example





2.2.1.2 Inbound Write Transaction

An inbound write transaction is initiated by a PCI master and is targeted at either 81341 and 81342 local memory or a 81341 and 81342 memory-mapped register.

Data flow for an inbound write transaction on the PCI bus is summarized as:

- The ATU claims the PCI write transaction when the PCI address is within the inbound translation window defined by the ATU Inbound Base Address Register (and Inbound Upper Base Address Register during DACs) and Inbound Limit Register.
- When the IWADQ has at least one address entry available and the IWQ has at least one buffer available, the address is captured and the first data phase is accepted.
- The PCI interface continues to accept write data until one of the following is true:
 - The initiator performs a disconnect.
 - The transaction crosses a buffer boundary.
- When an uncorrectable address error is detected during the address phase of the transaction, the uncorrectable address error mechanisms are used. Refer to [Section 2.7.1](#) for details of the uncorrectable address error response.
- When operating in the PCI-X mode when an uncorrectable attribute error is detected, the uncorrectable attribute error mechanism described in [Section 2.7.1](#) is used.
- When an uncorrectable data error is detected while accepting data, the slave interface sets the appropriate bits based on PCI specification. No other action is taken. Refer to [Section 2.7.3.6](#) for details of the inbound write uncorrectable data error response.

Once the PCI interface places a PCI address in the IWADQ, when IWQ has received data sufficient to cross a buffer boundary or the master disconnects on the PCI bus, the ATUs internal bus interface becomes aware of the inbound write. When there are additional write transactions ahead in the IWQ/IWADQ, the current transaction remains posted until ordering and priority have been satisfied (Refer to [Section 2.6.3](#)) and the transaction is attempted on the internal bus by the ATU internal master interface. The ATU does not insert target wait states nor do data merging on the PCI interface, when operating in the PCI mode.

In the PCI-X mode memory writes are always executed as immediate transactions, while configuration write transactions are processed as split transactions. The ATU generates a Split Completion Message, (with Message class = 0h - Write Completion Class and Message index = 00h - Write Completion Message) once a configuration write is successfully executed.

Also, when operating in the PCI-X mode a write sequence may contain multiple write transactions. The ATU handles such transactions as independent transactions.



Data flow for the inbound write transaction on the internal bus is summarized as:

- The ATU internal bus master requests the internal bus when IWADQ has at least one entry with associated data in the IWQ.
- When the internal bus is granted, the internal bus master interface initiates the write transaction by driving the translated address onto the internal bus. For details on inbound address translation, see [Section 2.2, "ATU Address Translation" on page 58](#).
- When an internal bus target does not claim write transaction, a master abort condition is signaled on the internal bus. The current transaction is flushed from the queue and **SERR#** may be asserted on the PCI interface.
- The ATU initiator interface attempts a 128-bit wide transfer on the internal bus. When the target that claims the request does not support 128-bit wide transfers, a 64-bit wide transfer is used. Transfers of use internal bus byte enables to mask the bytes not written in each data phase. Write data is transferred from the IWQ to the internal bus when data is available and the internal bus interface retains internal bus ownership. Refer to [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#) for details of internal bus operation.
- The internal bus interface stops transferring data from the current transaction to the internal bus when one of the following conditions becomes true:
 - The data from the current transaction has completed (satisfaction of byte count). An initiator termination is performed and the bus returns to idle.
 - A Master Abort is signaled on the internal bus. **SERR#** may be asserted on the PCI bus. Data is flushed from the IWQ.



2.2.1.3 Inbound Read Transaction

An inbound read transaction is initiated by a PCI initiator and is targeted at either 81341 and 81342 local memory or a 81341 and 81342 memory-mapped register space. The read transaction is propagated through the inbound transaction queue (ITQ) and read data is returned through the inbound read queue (IRQ).

When operating in the conventional PCI mode, all inbound read transactions are processed as delayed read transactions. When operating in the PCI-X mode, all inbound read transactions are processed as split transactions. The ATUs PCI interface claims the read transaction and forwards the read request through to the internal bus and returns the read data to the PCI bus. Data flow for an inbound read transaction on the PCI bus is summarized in the following statements:

- The ATU claims the PCI read transaction when the PCI address is within the inbound translation window defined by ATU Inbound Base Address Register (and Inbound Upper Base Address Register during DACs) and Inbound Limit Register.
- When operating in the conventional PCI mode, when the ITQ is currently holding transaction information from a previous delayed read, the current transaction information is compared to the previous transaction information (based on the setting of the DRC Alias bit in [Section 2.13.40, "ATU Configuration Register - ATUCR" on page 182](#)). When there is a match and the data is in the IRQ, return the data to the master on the PCI bus. When there is a match and the data is not available, a Retry is signaled with no other action taken. When there is not a match and when the ITQ has less than eight entries, capture the transaction information, signal a Retry and initiate a delayed transaction. When there is not a match and when the ITQ is full, then signal a Retry with no other action taken.
 - When an uncorrectable address error is detected, the uncorrectable address response defined in [Section 2.7](#) is used.
- When operating in the conventional PCI mode, once read data is driven onto the PCI bus from the IRQ, it continues until one of the following is true:
 - The initiator completes the PCI transaction. When there is data left unread in the IRQ, the data is flushed.
 - An internal bus Target Abort was detected. In this case, the Q-word associated with the Target Abort is never entered into the IRQ, and therefore is never returned.
 - Target Abort or a Disconnect with Data is returned in response to the Internal Bus Error.
 - The IRQ becomes empty. In this case, the PCI interface signals a Disconnect with data to the initiator on the last data word available.
- When operating in the PCI-X mode, when ITQ is not full, the PCI address, attribute and command are latched into the available ITQ and a Split Response Termination is signalled to the initiator.
- When operating in the PCI-X mode, when the transaction does not cross a 1024 byte aligned boundary, then the ATU waits until it receives the full byte count from the internal bus target before returning read data by generating the split completion transaction on the PCI-X bus. When the read requested crosses at least one 1024 byte boundary, then ATU completes the transfer by returning data in 1024 byte aligned chunks.



- When operating in the PCI-X mode, once a split completion transaction has started, it continues until one of the following is true:
 - The requester (now the target) generates a Retry Termination, or a Disconnection at Next ADB (when the requester is a bridge)
 - The byte count is satisfied.
 - An internal bus Target Abort was detected. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - internal bus target abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to [Section 2.7.1](#).
 - An internal bus Master Abort was detected. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=80h - Master abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to [Section 2.7.1](#)
- When operating in the conventional PCI mode, when the master inserts wait states on the PCI bus, the ATU PCI slave interface waits with no premature disconnects.
- When an uncorrectable data error occurs signified by **PERR#** asserted from the initiator, no action is taken by the target interface. Refer to [Section 2.7.3.5](#).
- When operating in the conventional PCI mode, when the read on the internal bus is target-aborted, either a target-abort or a disconnect with data is signaled to the initiator. This is based on the ATU ECC Target Abort Enable bit (bit 0 of the ATUIMR for ATU). When set, a target abort is used, when clear, a disconnect is used.
- When operating in the PCI-X mode, when the transaction on the internal bus resulted in a target abort, the ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - internal bus target abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to [Section 2.7.1](#).
- When operating in the conventional PCI mode, when the transaction on the internal bus resulted in a master abort, the ATU returns a target abort to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to [Section 2.7.1](#)
- When operating in the PCI-X mode, when the transaction on the internal bus resulted in a master abort, the ATU generates a Split Completion Message (message class=2h - completer error, and message index=80h - internal bus master abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to [Section 2.7.1](#).
- When operating in the PCI-X mode, when the Split Completion transaction completes with either Master-Abort or Target-Abort, the requester is indicating a failure condition that prevents it from accepting the completion it requested. In this case, since the Split Request addresses a location that has no read side effects, the ATU must discard the Split Completion and take no further action.



The data flow for an inbound read transaction on the internal bus is summarized in the following statements:

- The ATU internal bus master interface requests the internal bus when a PCI address appears in an ITQ and transaction ordering has been satisfied. When operating in the PCI-X mode the ATU does not use the information provided by the Relax Ordering Attribute bit. That is, ATU always uses conventional PCI ordering rules.
- Once the internal bus is granted, the internal bus master interface drives the translated address onto the bus. When a Retry is signaled, the request is repeated. When a master abort occurs, the transaction is considered complete and a target abort is loaded into the associated IRQ for return to the PCI initiator (transaction is flushed once the PCI master has been delivered the target abort).
- Once the translated address is on the bus and the transaction has been claimed, the internal bus target starts returning data using a split response. Read data is continuously received by the IRQ until one of the following is true:
 - The full byte count requested by the ATU read request is received. The internal bus completer's initiator interface performs an initiator completion in this case.
 - A partial byte count requested by the ATU read request is received. The completer's internal bus initiator interface performs an initiator completion in this case. Also, the completer reacquires the internal bus to deliver the remaining read data byte count to the ATU.
 - When operating in the conventional PCI mode, a Target Abort is received on the internal bus from the internal bus target. In this case, the transaction is aborted and the PCI side is informed.
 - When operating in the PCI-X mode, a Target Abort is received on the internal bus from the internal bus target. In this case, the transaction is aborted. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - internal bus target abort) on the PCI bus to inform the requester about the abnormal condition. The ITQ for this transaction is flushed.

To support *PCI Local Bus Specification*, Revision 2.0 devices, the ATU can be programmed to ignore the memory read command (Memory Read, Memory Read Line, and Memory Read Multiple) when trying to match the current inbound read transaction with data in a DRC queue which was read previously (DRC on target bus). When the Read Command Alias Bit in the ATUCR register is set, the ATU does not distinguish the read commands on transactions. For example, the ATU enqueues a DRR with a Memory Read Multiple command and performs the read on the internal bus. Some time later, a PCI master attempts a Memory Read with the same address as the previous Memory Read Multiple. When the Read Command Bit is set, the ATU would return the read data from the DRC queue and consider the Delayed Read transaction complete. When the Read Command bit in the ATUCR was clear, the ATU would not return data since the PCI read commands did not match, only the address.



2.2.1.4 Inbound Configuration Cycle Translation

The 81341 and 81342 ATU only accepts Type 0 configuration cycles with a function number of zero when bit[7] of the ATUHTR (see [Section 2.13.11, "ATU Header Type Register - ATUHTR" on page 158](#)) is cleared or function numbers of zero and one when bit[7] of the ATUHTR is set.

The ATU is configured through the PCI bus. When operating in conventional PCI mode, all inbound configuration cycles are processed as delayed transactions. When operating in PCI-X mode, all inbound configuration cycles are processed as split transactions. The translation mechanism for inbound configuration cycles is defined by the *PCI Local Bus Specification*, Revision 2.3.

The ATU configuration space is selected by a PCI configuration command and claims access (by asserting **P_DEVSEL#**) when the **P_IDSEL** pin is asserted, the PCI command indicates a configuration read or write, and address bits **P_AD[1:0]** are 00₂ all during the address phase. The ATU interface ignores any configuration command (**P_IDSEL** active) where **P_AD[1:0]** are not 00₂ (e.g. Type 1 commands). During the configuration access address phase, the PCI address is divided into a number of fields to determine the actual configuration register access. These fields, in combination with the byte enables during the data phase create the unique encoding necessary to access the individual registers of the configuration address space:

- **P_AD[7:2]** - Register Number. Selects one of 64 DWORD registers in the ATU PCI configuration address space.
- **P_C/BE[3:0]#** - Used in data phase. Selects which actual configuration register is used within the DWORD address. Creates byte addressability of the register space.
- **P_AD[10:8]** - Function Number. Used to select which function of a multi-function device is being accessed. The ATU is function 0 and therefore it only responds to 000₂ in this bit field and ignore all other bit combinations.
- **P_AD[27:24]** - Upper Register Number. In PCI-X Mode 2, Upper Register Number and Lower Register Number combine to select one of 1024 DWORD registers in the ATU PCI configuration address space.

Note: In PCI-X Mode 2, the ATU does not support any extended capabilities list items starting at offset 100H indicated by a Null Enhanced Capability Header at offset 100H (i.e., Enhanced Capability Header with a Capability ID of 0000H, a Capability Version of 0H, and a Next Capability offset of 000H).



ATU configuration address space starts at internal address 3100H. Therefore, **P_AD[7:2]** equal to 000000₂ equates to address 3100H and **P_AD[7:2]** equal to 000001₂ results in address 3104H and so on.

For inbound configuration reads, IRQ and ITQ are used in the same manner as inbound memory read operations. The internal bus cycle that results are a 32-bit transaction.

For inbound configuration writes, ATU adds a delayed write data queue (IDWQ), which holds data the same way as the IWQ. Transaction information from the configuration write operation on the PCI interface is captured into the IDWQ (when full, a Retry is signaled). Data from delayed write (split write in PCI-X mode) request cycle is latched into IDWQ and forwarded to the internal bus interface. Once transaction ordering and priority have been satisfied, the internal bus master interface requests the internal bus and delivers write data to the target as defined in [Section 2.2.1.2](#).

Status of the internal bus transaction is returned to the PCI bus, PCI initiator. When operating in conventional PCI mode, the initiator retry cycle is accepted once the write has been completed on the internal bus and status has been captured for return to the PCI master. When operating in PCI-X mode, a Split Completion Message (message class=0h and message index= 00h - Normal Completion) is generated on the PCI bus, once the write has been completed on the internal bus and status has been latched for return to the PCI master. Since Master Aborts and Target Aborts cannot occur during internal configuration cycles, normal completion status is returned. Data from PCI completion transaction is discarded.



2.2.1.5 Discard Timers

The ATU implements discard timers for inbound delayed transactions. These timers prevent deadlocks when the initiator of a retried delayed transaction fails to complete the transaction within 2^{10} or 2^{15} PCI clock cycles on the initiating bus when operating in the conventional PCI mode. The timer starts counting when the delayed request becomes a delayed completion by completing on the internal bus and all passing rules are satisfied. When the originating master on the PCI bus has not retried the transaction before the timer expires, the completion transaction is discarded.

Discard timer values are controlled by the ATU Configuration Register's Discard Timer Value bit. The ATU queues covered by discard timers are the IRQ and the IDWQ. After discarding a transaction, the ATU must set the Discard Timer Status bit in the ATU Configuration Register. The ATU does *not* assert the **SERR#** signal after discarding a transaction.



2.2.2 Outbound Transactions- Single Address Cycle (SAC) Internal Bus Transactions

Outbound transactions initiated by the 81341 and 81342 core processor are directed to the PCI interface through the ATU. The core processor always generates Single Address Cycles on the internal bus. As a PCI master, the ATU is capable of PCI I/O transactions, PCI memory reads (in case of conventional PCI), memory read DWORD (in case of PCI-X), PCI memory writes, configuration reads and writes, and DAC cycles. Outbound memory transactions are always attempted as 64-bit PCI transactions. Outbound memory write operations are performed as posted operations and outbound memory read operations are all performed as split read operations.

Outbound transactions use a separate set of queues from inbound transactions. Outbound write operations have their address entered into the outbound write address queue (OWADQ) and their data into the outbound write queue (OWQ). Outbound read transactions, performed as split transactions, use the Outbound Transaction Queue (OTQ) to store address, and get data returned into the outbound read queue (ORQ). Refer to [Section 2.6.2](#) for details of outbound queue architecture. Outbound configuration transactions use a special outbound port structure. Refer to [Section 2.2.3](#) for details.

For outbound write transactions, the ATU is a target on the internal bus and initiator on the PCI bus. For outbound read transactions, the ATU is a completer on the internal bus (initially accepts the split read as a target and then provides read data by initiating a split completion). Internal bus operation is defined in [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#). ATU specific internal bus operation is defined in [Section 2.2.6](#).

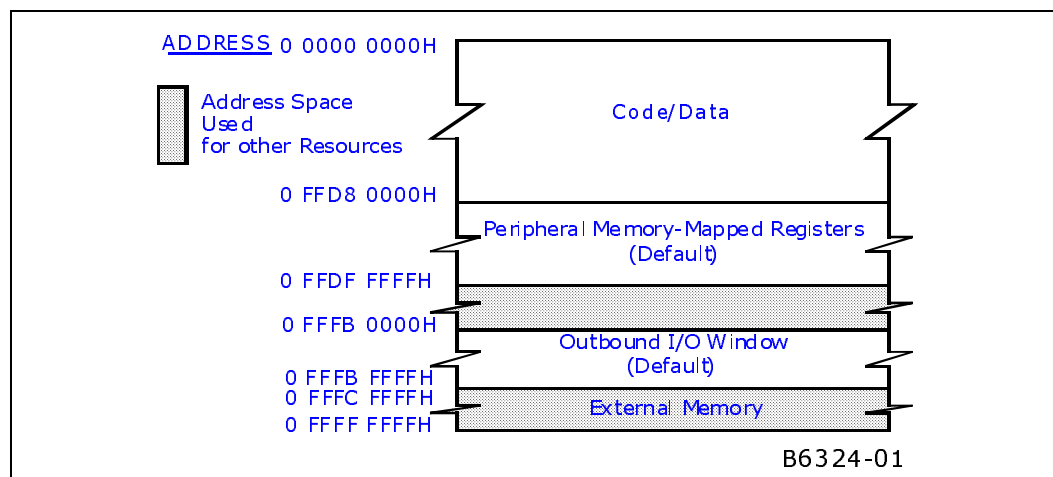


2.2.2.1 Outbound Address Translation - Internal Bus Transactions

In addition to providing the mechanism for inbound translation, the ATU translates Intel XScale® processor-initiated cycles to the PCI bus. This is known as *outbound address translation*. Outbound transactions are processor or ADMA transactions targeted at the PCI bus. The ATU internal bus target interface claims internal bus cycles and completes the cycle on the PCI bus on behalf of the Intel XScale® processor or ADMA.

Figure 7 shows 4 Gbyte memory section 0 (Internal Bus Address [35:32] = 0000₂) of the 81341 and 81342 memory map with all reserved address locations highlighted. By default, the 64KByte outbound I/O window is from 0.FFFB.0000H to 0.FFFB.FFFFH while the PMMR registers, by default, reside from 0.FFD8.0000H to 0.FFDF.FFFFH.

Figure 7. 4 Gbyte Section 0 of the Internal Bus Memory Map



By default, Outbound Memory Window 0, Outbound Memory Window 1, Outbound Memory Window 2, and Outbound Memory Window 3 reside in 4 Gbyte memory sections 1, 2, 3 and 4 respectively, of the 64 Gbyte Internal Bus address space.

The ATU response to Outbound Transactions is globally controlled by the ATU Configuration Register Outbound Enable bit, as well as the Bus Master Enable bit in each function. When the Outbound Enable bit is deasserted, the internal bus outbound transaction master-abort are not forwarded to the PCI Express Domain. When the Outbound Enable bit is asserted, the relevant Bus Master Enable bit for each function determines appropriate response to an outbound transaction.

Table 2 describes the Outbound ATUs behavior for the different combinations of these control bits.

Table 2. Outbound Address Translation Control

Outbound Response	Outbound Enable ^a (ATUCR[1])	Bus Master Enable ^b
Master-Abort	0	0
Master-Abort	0	1
Retry	1	0
Claim ^c	1	1

a. In addition, the outbound memory windows need to be individually enabled in order to claim. When, disabled, the outbound memory windows does not claim. By default, Outbound Memory Window 0, and Outbound Memory Window 1 are enabled. Outbound Memory Windows 2 and 3 are disabled by default.

b. In a multi-function configuration, each function independently controls its own Bus Master Enable bit.

c. The ATU may respond with a Retry in this case when the Outbound Transaction Queues are full.

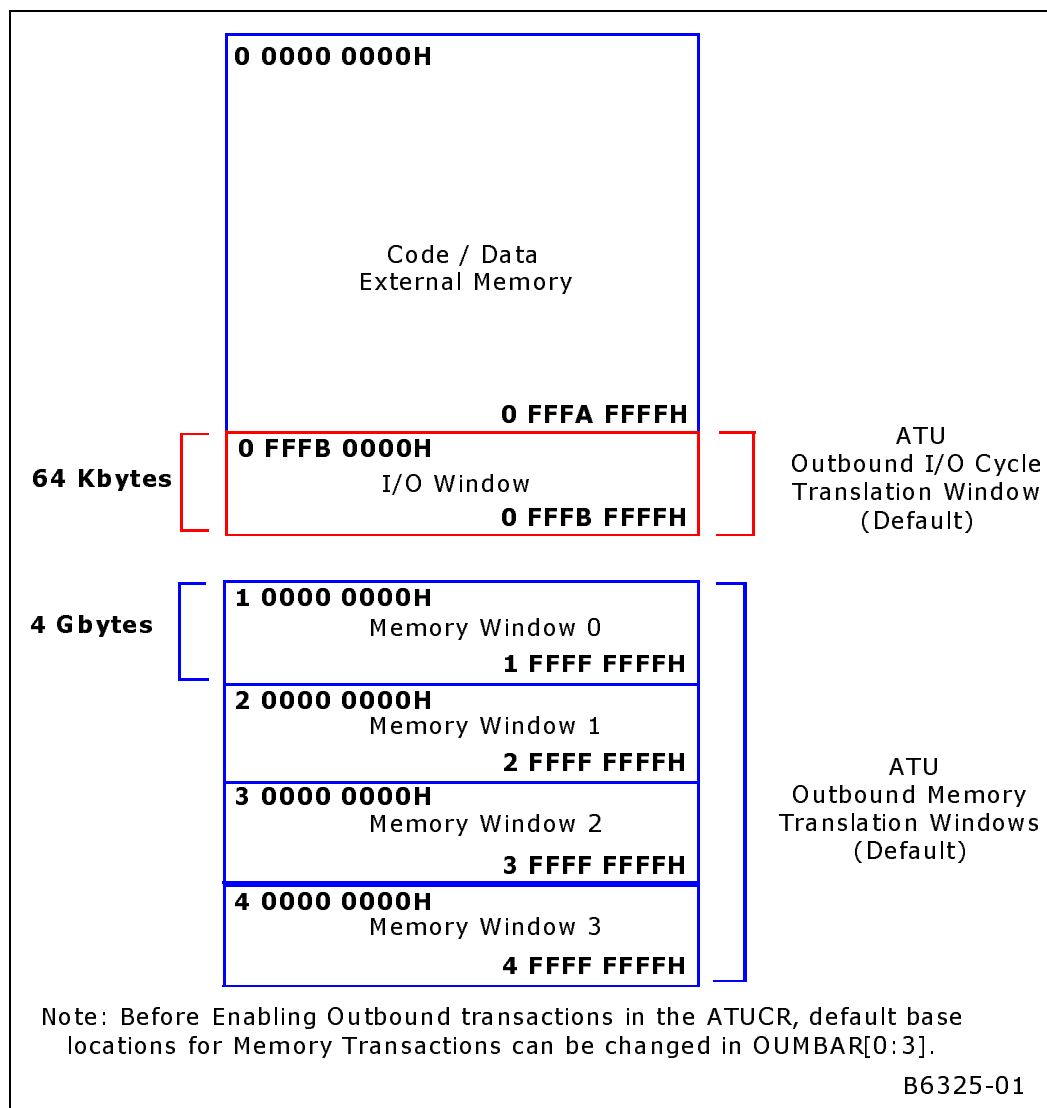


2.2.2.2 Outbound Address Translation Windows

Inbound translation involves a programmable inbound translation window consisting of a base and limit register and a value register for PCI to internal bus translation. The outbound address translation windows use a similar methodology except that the outbound translation window limit sizes are fixed in the 81341 and 81342 internal bus address space; this removes the need for separate limit registers.

Figure 8 on page 74 illustrates the five outbound address translation windows.

Figure 8. Outbound Address Translation Windows





ATU has four 4 Gbyte outbound memory translation windows and one 64 Kbyte outbound I/O translation window. By default, Outbound Memory Window 0 (OUMBAR0), Outbound Memory Window 1 (OUMBAR1), Outbound Memory Window 2 (OUMBAR2), and Outbound Memory Window 3 (OUMBAR3) reside in 4 Gbyte memory sections 1, 2, 3, and 4, respectively. The default location of the 64 Kbyte outbound I/O window range is from 0.FFFB.0000H to 0.FFFB.FFFFH. The following registers are used to specify the five 4 Gbyte windows for claiming Outbound Memory transactions:

- Outbound Upper Memory Base Address Register 0 (OUMBAR0)
 - Default Value equal to 01H.
- Outbound Upper Memory Base Address Register 1 (OUMBAR1)
 - Default Value equal to 02H.
- Outbound Upper Memory Base Address Register 2 (OUMBAR2)
 - Default Value equal to 03H.
- Outbound Upper Memory Base Address Register 3 (OUMBAR3)
 - Default Value equal to 04H.
- Outbound I/O Base Address Register (OIOBAR)
 - Default Value equal to 0FFF B000H

An internal bus cycle with an address within one outbound window initiates a read or write cycle on the PCI bus. The PCI cycle type depends on which translation window the local bus cycle “hits”. The read or write decision is based on the internal bus cycle type.

ATU has windows dedicated to the following outbound PCI/PCI-X transaction types:

- Memory reads and Memory writes - Memory Window
- I/O reads and writes - I/O Window

Table 3. Internal Bus-to-PCI Command Translation for Memory Windows

Internal Bus Command	Conventional PCI Command	PCI-X Command
Write ^a	Memory Write	Memory Write
Write ^b	Memory Write and Invalidate ^c (DMA) Memory Write (Memory Window 0-3)	Memory Write Block (DMA) Memory Write (Memory Window 0-3)
Read ^d	Memory Read	Memory Read DWORD
Read ^e	Memory Read Multiple	Memory Read Block

a. The internal bus request does **not** cross a QWORD address boundary.

b. The internal bus request does cross a QWORD address boundary

c. The ATU converts Write to Memory Write and Invalidate when the following four conditions are met, otherwise the Write is converted to a Memory Write:

- 1.) Memory Write and Invalidate transactions are enabled in the “ATU Command Register - ATUCMD”.
- 2.) The Cache Line size is set to 8 or 16 DWORDS in the “ATU Cacheline Size Register - ATUCLSR”.
- 3.) Starting address of the Outbound PCI bus request is aligned to the cache line size.
- 4.) Byte count intended for the Outbound PCI bus request is a multiple of the cache line size.

d. The internal bus request does **not** cross a DWORD address boundary.

e. The internal bus request does cross a DWORD address boundary

Table 4. Internal Bus-to-PCI Command Translation for I/O Window

Internal Bus Command ^a	Conventional PCI Command	PCI-X Command
Write	I/O Write	I/O Write
Read	I/O Read	I/O Read

a. User should designate memory region containing I/O Window as non-cacheable and non-bufferable from Intel XScale® processor. This insures all load/stores to I/O Window are of DWORD quantities. In the event that the user inadvertently issues a read to the I/O Window which crosses a DWORD address boundary, the ATU target aborts the transaction. Only bytes 3:0 are relevant dependent on the Byte Enables.



The translation portion of outbound ATU transactions is accomplished with a value register in the same manner as inbound translations. Each outbound memory window is associated with one translation register which provides the upper translation addresses (OUMWVR0-3). When the corresponding OUMWVRx register is all-zero a SAC transaction is generated on the PCI bus. Otherwise, a DAC is generated on the PCI bus using the value in the OUMWVRx register for the upper 32-bit address. ATU uses the following registers during outbound address translation:

- Outbound Upper 32-bit Memory Window Value Register 0 (OUMWVR0)
- Outbound Upper 32-bit Memory Window Value Register 1 (OUMWVR1)
- Outbound Upper 32-bit Memory Window Value Register 0 (OUMWVR2)
- Outbound Upper 32-bit Memory Window Value Register 1 (OUMWVR3)
- Outbound I/O Window Value Register (OIOWVR)
- Outbound Configuration Cycle Address Register (OCCAR)

See [Section 2.13](#) for details on outbound translation register definition and programming constraints.

The translation algorithm used, as stated, is very similar to inbound translation. For memory transactions, the algorithm is:

Equation 3. Outbound Address Translation

$$\text{PCI Address} = (\text{Internal\_Bus\_Address} \& 0.\text{FFFF.FFFFH}) \mid (\text{Upper\_Window\_Value\_Register} \ll 32)$$

For memory transactions, the internal bus address is bitwise ANDed with the inverse of 4 Gbytes which clears the upper 4 bits of the 36 bit address. The result is bitwise ORed with the outbound upper window value register left shifted by 32 to create the Upper 32-bits of the PCI address. When the Upper 32-bits of the PCI Address equals 0000 0000H, the ATU generates a SAC transaction on the PCI bus, otherwise, a DAC transaction is used.

For I/O transactions, the algorithm is:

Equation 4. I/O Transactions

$$\text{PCI Address} = (\text{Internal\_Bus\_Address} \& 0.0000.\text{FFFFH}) \mid \text{Window\_Value\_Register}$$

For I/O transactions, the internal bus address is bitwise ANDed with the inverse of 64 Kbytes which clears the upper 20 bits of address. Address aliasing is prevented by the outbound window value registers which only allow values on boundaries equivalent to the window's length.

2.2.2.3 Outbound DMA Transactions

The ATU provides all ADMA channels with a transparent path through to the External PCI bus. The entire 64-bit Host I/O Interface address programmed in the DMA descriptor is passed to the PCI bus unmodified.

Depending on the contents of the Host / XOR Destination Upper Address word of a particular ADMA descriptor, the ATU may initiate a SAC or a DAC transaction on the PCI bus in response to the ADMA channel's request (see [Section 5.3, "ADMA Descriptors"](#) on page 466 for more details).



2.2.3 Outbound Write Transaction

An outbound write transaction is initiated by the Intel XScale® processor² or by one of the ADMA channels and is targeted at a PCI target on the PCI bus. The outbound write address and write data are propagated from the 81341 and 81342 internal bus to a PCI bus through OWADQ and OWQ, respectively.

The ATUs internal bus target interface claims the write transaction and forwards the write data through to the targeted PCI bus. The data flow for an outbound write transaction on the internal bus is summarized in the following statements:

- For Single Address Cycles (SACs), ATU internal bus target interface latches the address from the internal bus into the OWADQ when that address is inside one of the outbound translate windows (see [Section 2.6](#)) and the OWQ is not full.
- For Dual Address Cycles (DACs), ATU internal bus target interface latches address from the internal bus into the OWADQ when the OWQ is not full and OWADQ is not full.
- Once outbound address is latched, internal bus target interface stores write data into the OWQ until the internal bus transaction completes or the reaches a buffer boundary. The initiator of the transaction is disconnected at an ADB when the transaction reaches a buffer boundary.
- When the OWADQ is full, the target interface signals a Retry on the internal bus to the outbound cycle initiator.
- When OWADQ latches the address and corresponding data is latched in a buffer in OWQ, the outbound cycle is enabled for transmission on the PCI Bus and PCI interface requests PCI bus.

-
2. For best performance, the user should designate the two Outbound Memory Windows as non-cacheable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 12, "ATU Outbound Data Flow Ordering Rules" on page 93](#). In the event that the user requires strict ordering to be maintained. In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cacheable/non-bufferable and enforce the requirement in software.



The PCI interface is responsible for completing the outbound write transaction with the PCI address translated from the OWADQ and the data in the OWQ. The data flow for an outbound write transaction on the PCI bus is summarized in the following statements:

- ATU PCI interface requests PCI bus, when completed internal bus transaction is in OWADQ and data associated with transfer in OWQ. Once bus is granted, PCI master interface writes PCI translated address from OWADQ to PCI bus and waits for transaction to be claimed.
- When Master Abort seen during address phase, transaction flushed and OWADQ/OWQ are cleared. [Section 2.7.5](#) has full details on PCI master abort conditions during outbound transactions.
- In conventional PCI mode, once PCI write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
 - PCI target signals a Retry or Disconnect. The ATU PCI master attempts to reacquire the PCI bus to complete the write transaction.
 - **GNT#** signal is deasserted and master latency timer has expired. In this case, master interface attempts to reacquire PCI bus and complete write transaction.
 - PCI target signals a Target-Abort. In this case, OWQ and OTQ are cleared and transaction aborted. Appropriate error bits are set as defined in [Section 2.7.6](#).
 - Transaction terminates normally by transferring all data (full byte count) associated with it. The write address is removed from the OWADQ and the interface returns to idle.
- In PCI-X mode, once the PCI memory write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
 - The PCI target signals a Retry or Single Data Phase Disconnect. The ATU PCI initiator attempts to reacquire the PCI bus to complete the write transaction.
 - Reacquire the PCI bus to complete the write transaction.
 - **GNT#** signal is deasserted and the master latency timer has expired. In this case, the master interface attempts to reacquire the PCI bus and complete the write transaction.
 - PCI target signals a Target-Abort. In this case, the OWQ and OWADQ are cleared and the transaction is aborted. The appropriate error bits are set as defined in [Section 2.7.6](#).
 - Transaction terminates normally with Satisfaction of Byte Count. The write address is removed from the OWADQ and the interface returns to idle.
- In the PCI-X mode, once the PCI I/O write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
 - PCI target signals a Retry. The ATU PCI initiator attempts to reacquire the PCI bus to complete the write transaction.
 - Transaction terminates normally with Satisfaction of Byte Count or with Single Data Phase Disconnect. The write address is removed from the OWADQ and the interface returns to idle.
 - PCI target signals a Target-Abort. In this case, the OWQ and OWADQ are cleared and the transaction is aborted. The appropriate error bits are set as defined in [Section 2.7.6](#).
 - Transaction terminates with Split Response Termination. The write address is removed from the OWADQ and the interface returns to idle only when it receives the corresponding Split Completion Message.

When an uncorrectable data error is encountered (**PERR#** detected), the master interface continues writing data to clear the queue.

In the conventional PCI mode when the PCI target deasserts **TRDY#**, no action is taken by the ATU master other than inserting wait states.



2.2.4 Outbound Read Transaction

An outbound read transaction is initiated by the Intel XScale® processor³ or an ADMA channel and is targeted at a PCI slave on the PCI bus. The read transaction is propagated through the outbound transaction queue (OTQ) and read data is returned through the outbound read queue (ORQ).

The ATUs internal bus target interface claims the Memory Read Dword and Memory Read Block and Alias to Memory Read Block transaction and forwards the read request through to the PCI bus and returns the read data to the internal bus.

The data flow for an outbound read transaction on the internal bus is summarized in the following statements:

- For Single Address Cycles (SACs), the ATU internal bus interface latches the internal bus address when the address is inside an outbound address translation window and OTQ is not full. For Dual Address Cycles (DACs), ATU internal bus target interface latches the address from the internal bus into OTQ irrespective of the address. All read transactions are handled as split transactions. When OTQ is full (previous outbound transactions in progress), the internal bus interface signals a Retry to the transaction initiator.
- When during the completion cycle on the PCI interface, a master abort is encountered, a flag is set and the ATU aborts the completion to the internal bus requester. The OTQ is cleared of the transaction.
- When during the completion cycle on the PCI interface, a target abort is encountered, a flag is set and the ATU aborts the completion to the internal bus requester. The OTQ is cleared of the transaction.
- Once the transaction completes on the PCI bus, the ATU generates a split completion transaction to return data to the internal bus requester.
- When operating in the PCI-X mode, ATU may receive a split completion error message when attempting to read data on the PCI bus. In this case, ATU notifies the internal bus requester about the error by aborting the completion to the internal bus requester. The OTQ is cleared of the transaction.

The data flow for an outbound read transaction on the PCI bus is summarized in the following statements:

- The ATU PCI interface requests the PCI bus when the head of the OTQ has at least one entry and the ordering rules are satisfied. Once the bus is granted, the PCI interface transfers the PCI translated address from the OTQ to the PCI bus and wait for the transaction to be claimed.
- When no **DEVSEL#** is asserted, a master abort is signaled. This is passed through to the internal bus target interface.
- When a target abort is signaled from the PCI target, the target abort is returned to the internal bus and the PCI interface returns to idle.
- When operating in the PCI-X mode the read transaction may terminate as a split response termination. Then the ATU receives data during the corresponding split completion transaction. When an error occurs, the ATU may receive a split completion error message.

-
3. For best performance, designate the two Outbound Memory Windows as non-cacheable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of ATU outbound queue architecture. However, be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 12, "ATU Outbound Data Flow Ordering Rules" on page 93](#). In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cacheable/non-bufferable and enforce the requirement in software.



2.2.5 Outbound Configuration Cycle Translation

Outbound ATU provides a port programming model for outbound configuration cycles.

Performing an outbound configuration cycle to the PCI bus involves up to two internal bus cycles:

1. Writing Outbound Configuration Cycle Address Register (OCCAR) with PCI address used during configuration cycle. See the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 for information regarding configuration address cycle formats. This IB bus cycle enables the transaction.
2. Writing or reading Outbound Configuration Cycle Data Register (OCCDR). A read causes a configuration cycle read to the PCI bus with the address in the outbound configuration cycle address register. Note that the Internal Bus read is executed as a split transaction. Similarly, a write initiates a configuration cycle write to PCI with the write data from the second processor cycle. ⁴ Configuration cycles are non-burst and restricted to a single 32-bit word cycle.

When the Configuration Cycle Data Register is written, data is latched and forwarded to the PCI bus with the internal target issuing a single data phase disconnect with 32-bit data only. This cycle does not receive an **ACK64#** from the ATU and therefore is defined as 32-bit only.

Note, the programming model uses the register interface for outbound configuration cycles, from a hardware standpoint, the address is entered into OTQ (reads) or OWADQ (writes), configuration write data goes through OWQ and configuration read data is returned in the ORQ.

Note: Outbound configuration cycle data registers are not physical registers. They are 81341 and 81342 memory mapped addresses used to initiate a transaction with the address in the associated address register.

2.2.5.1 PCI-X Mode 1 Considerations for Outbound Configuration Cycles

Configuration cycle address Bits 15:11 for Type 0 configuration cycles are defined differently for Conventional versus PCI-X modes. When 81341 and 81342 software programs OCCAR to initiate a Type 0 configuration cycle, always load OCCAR based on the PCI-X definition for Type 0 configuration cycle address. In Conventional mode, 81341 and 81342 clears OCCAR bits 15:11 prior to initiating an outbound Type 0 configuration cycle.

During the attribute phase of a Type 0 configuration transaction, the Secondary Bus Number field (bits 7:0) is set equal to the Requester Bus Number (bits 15:8 of the "PCI-X Status Register - PCIXSR" on page 198).

-
4. The designate the memory region containing OCCDR as non-cacheable and non-bufferable from the Intel XScale® processor. This insures that all load/stores to OCCDR are only of DWORD quantities. In event the user inadvertently issues a read to OCCDR that crosses a DWORD address boundary, the ATU target aborts the transaction. All writes are terminated with a Single-Phase-Disconnect and only bytes 3:0 is relevant.



2.2.5.2 PCI-X Mode 2 Considerations for Outbound Configuration Cycles

In addition to the PCI-X Mode 1 changes relative to Conventional PCI mode, for PCI-X Mode 2, the definition for bits 31:24 of the configuration address has changed. Bits 31:28 are Reserved while bits 27:24 represent the enhanced configuration cycle upper register address providing up to 4 Kbytes of configuration register space. In addition, a consequence of this change is that Device Numbers 15:8 are not longer available. For Mode 2 implementations, the System hardware is restricted to Device Numbers 7:0. Software needs to read the PCSR to confirm that the PCI interface is operating in Mode 2 in order to set bits 27:24 properly.

2.2.5.3 Outbound Configuration Cycle Error Conditions

Master aborts during outbound configuration reads result in ATU aborting the read completion on the internal bus.

Target aborts during outbound configuration reads result in ATU aborting the read completion on the internal bus.

Uncorrectable errors during outbound configuration reads result in ATU aborting the read completion on the internal bus.

Uncorrectable errors detected by target of an outbound configuration write may result in the ATU receiving either of the two Split Completion Write Uncorrectable Data Error Messages (with message class=2h - completer error and message index=01h - split write uncorrectable data error or with message class=1h - bridge error and message index=02h - write uncorrectable data error) on the PCI bus. When Parity Checking is enabled, the ATU sets error bits in the ATUSR and the PCIXSR. The Intel XScale® processor is interrupted when the Split Completion Error and/or Master Data Parity interrupt(s) are unmasked.



2.2.6 Internal Bus Operation

Complete internal bus operation of the 81341 and 81342 is defined in [Chapter 6.0, “System Controller \(SC\) and Internal Bus Bridge”](#). The ATU acts as both internal bus master and internal bus slave device. A summary of ATU internal bus specific operation follows for both master and slave operation



2.3 Big Endian Byte Swapping

Each memory and I/O window has an associated byte swapping enable located in the following address translation registers:

- bit 0 of Inbound Address Translate Value Register 0-3 (IATVR0-3)
- bit 0 of Inbound Expansion ROM Translate Value Register (ERTVR)
- bit 0 of Outbound I/O Window Translate Value Register (OIOWTVR)
- bit 27 of Outbound Upper Memory BAR 0-3 (OUMBAR0-3)

Note: The Messaging Unit (MU) Memory is mapped in PCI Window 0 (ATU Base Address Register 0) along with the MSI-X table structures. Byte swapping should not be enabled for BAR0 when using MSI-X.

2.3.1 Inbound Byte Swapping

When enabled, the swapping occurs as described in [Figure 9, “Inbound Byte Swapping for 32-bit PCI” on page 83](#) and [Figure 10, “Inbound Byte Swapping for 64-bit PCI” on page 83](#). The bytes are swapped within a DWORD and byte swapping is performed for all transactions regardless of byte count.

Figure 9. Inbound Byte Swapping for 32-bit PCI

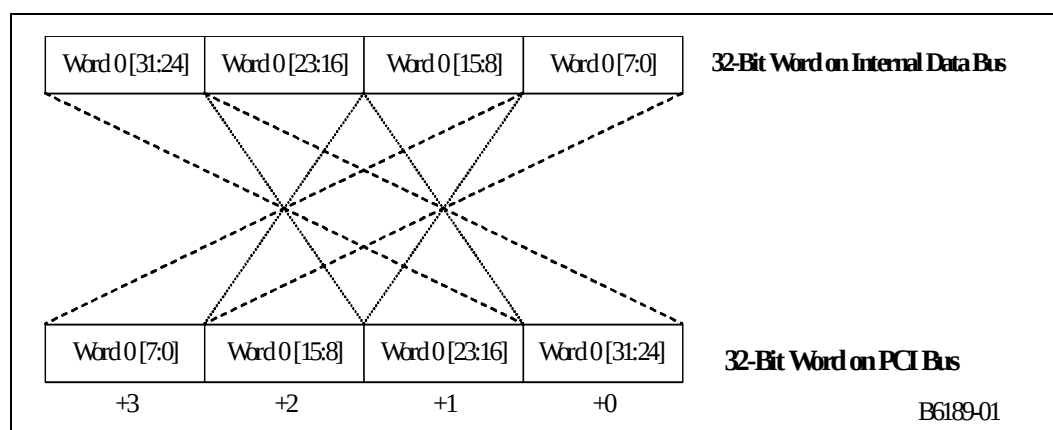
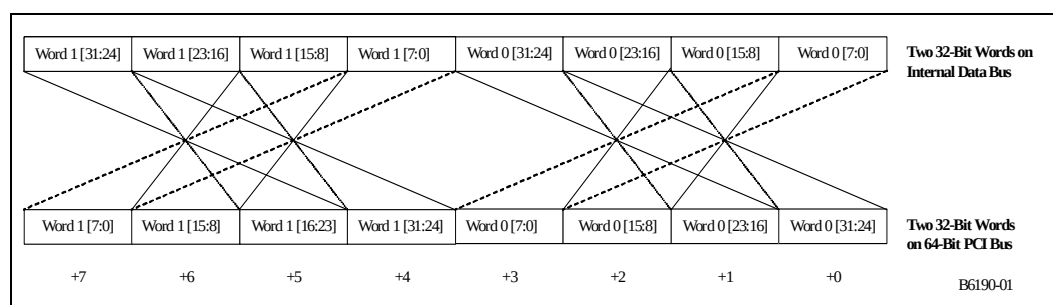


Figure 10. Inbound Byte Swapping for 64-bit PCI



2.3.2 Outbound Byte Swapping

When enabled, the swapping occurs as described in Figure 11, “Outbound Byte Swapping for Transaction with Byte Count of 1” on page 84, Figure 12, “Outbound Byte Swapping for Transaction with Byte Count of 2” on page 84, and Figure 13, “Outbound Byte Swapping for Transaction with Byte Count of 3 or Larger” on page 84. The bytes are swapped within a 32-bit DWORD and the type of byte swapping performed is determined by the transaction byte count. For Byte Count of 3 or larger transactions, no byte swapping is performed.

Note: The byte swapping capability of the ADMA unit should be used to swap bytes within each DWORD for PCI-to-Memory Read/Write DMA transfers.

Figure 11. Outbound Byte Swapping for Transaction with Byte Count of 1

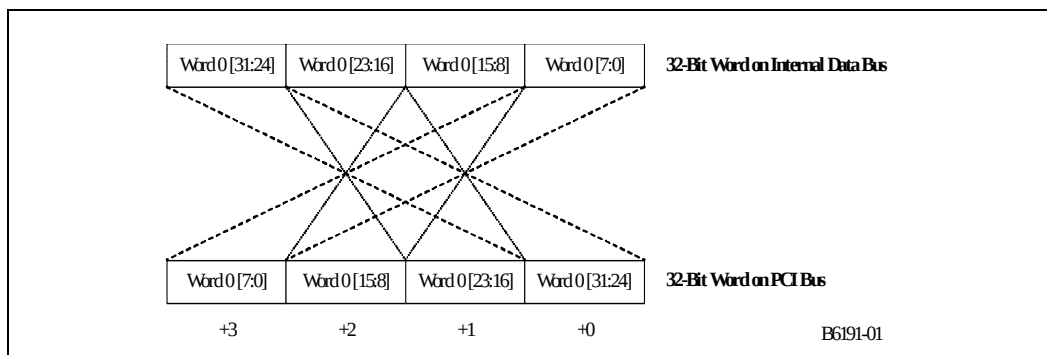


Figure 12. Outbound Byte Swapping for Transaction with Byte Count of 2

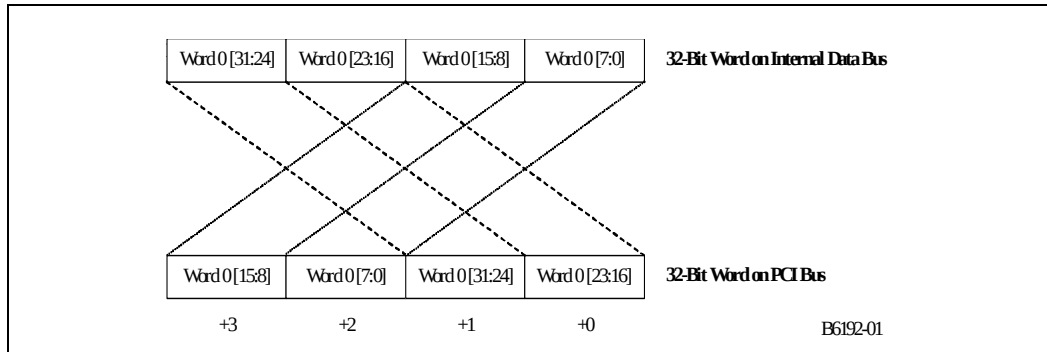
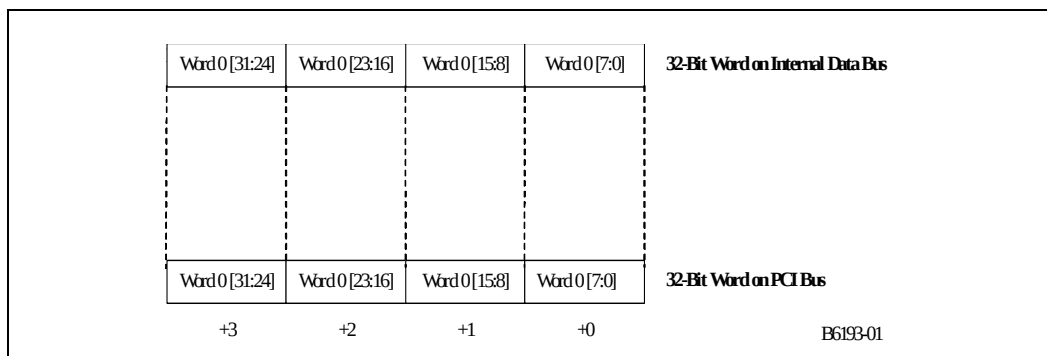


Figure 13. Outbound Byte Swapping for Transaction with Byte Count of 3 or Larger





2.4 CompactPCI Hot-Swap

The 81341 and 81342 meets the standard requirements to be considered “Hot-Swap Silicon” detailed in the *Compact PCI Hot-Swap Specification*, Revision 2.1. This includes a dedicated pin interface and extended capability header.

Hot-Swap Control and Status Register is implemented via the Extended Capability Pointer mechanism in the ATUs configuration space ([Section 2.13.61, “CompactPCI Hot-Swap Capability ID Register”](#)).

Provides Software Connection Control Resources for ENUM#, Hot-Swap Switch, blue LED and Device Hiding.

Handle Switch de-bouncing is implemented.

See [Section 2.4.1.1, “Compact PCI Hot-Swap Mode Select”](#) for a description of **HS_SM#** and HS_FREQ[1:0] requirements.

2.4.1 Pin Interface

Table 5. Compact PCI Hot-Swap

Signal	Width	Type	Description
HS_ENUM#	1	Od-O	Compact PCI Hot-Swap Event: Conditionally asserted to notify the system host that either a board has been freshly inserted or is about to be extracted. This signal informs the system host that the configuration of the system has changed. The system host then performs any necessary maintenance such as installing or quiescing a device driver.
HS_LSTAT	1	I	Compact PCI Hot-Swap Latch Status: An input indicating the state of the ejector switch. 0 = Indicates the ejector switch is closed. 1 = Indicates the ejector switch is open. When Compact PCI Hot-Swap is not supported, this signal must be tied low.
HS_LED_OUT	1	O	LED Output: 81341 and 81342 outputs a logic one to illuminate the Hot-Swap LED.
HS_SM#	1	I	Hot-Swap Startup Mode (Strap): This strap is sampled at the rising edge of P_RST# to indicate Hot-Swap Mode. When 1b and Configuration Retry Mode enabled (bit 2 of the PCSR), indicates 81341 and 81342 retries all Configuration transactions. When 0b and Configuration Retry Mode enabled (bit 2 of the PCSR), indicates 81341 and 81342 does not claim Configuration transactions and the bus mode is determined by the HS_FREQ[1:0] straps.
HS_FREQ[1:0]	2	I	This pins are reserved for determining Bus frequency and mode during a PCI-X Hot-Swap event and are only valid when HS_SM# = 0. The bus frequency and mode are described below: 11 => PCI Mode, 33 or 66 MHz. Use P_M66EN to determine frequency 10 => PCI-X Mode, 66 MHz 01 => PCI-X Mode, 100 MHz 00 => PCI-X Mode, 133 MHz These pins are not used for non-Hot-Swap systems.
Total	6		



2.4.1.1 Compact PCI Hot-Swap Mode Select

HS_SM# must be asserted (0b) to enable Hot-Swap functionality.

HS_FREQ[1:0] pins allow the 81341 and 81342 to determine the cPCI backplane operating frequency without needing to see a PCI-X initialization pattern. These pins are only valid when **HS_SM#** is sampled as 0b during P_RST#.

Table 6. HS_FREQ Encoding ^a

HS_FREQ[1:0]	P_M66EN	Operating Mode	Bus Frequency	PCSR[19:16]
11	0	PCI	33 MHz	1111
11	1	PCI	66 MHz	1111
10	-	PCI-X (Mode 1)	66 MHz	1110
01	-	PCI-X (Mode 1)	100 MHz	1101
00	-	PCI-X (Mode 1)	133 MHz	1100

a. Hot-Swap is not supported in PCI-X Mode 2.



2.5 Expansion ROM Translation Unit

The inbound ATU supports one address range (defined by a base/limit register pair) used for the Expansion ROM. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details on Expansion ROM format and usage.

During a powerup sequence, initialization code from Expansion ROM is executed once by the host processor to initialize the associated device. The code can be discarded once executed. Expansion ROM registers are described in [Section 2.13.37](#) through [Section 2.13.39](#).

The inbound ATU supports an inbound Expansion ROM window which works like the inbound translation window. A read from the expansion ROM windows is forwarded to the internal bus. The address translation algorithm is the same as the inbound translation; see [Section 2.2.1.1, "Inbound Address Translation" on page 61](#). As a PCI target, the Expansion ROM interface behaves as a standard ATU interface and is capable of returning a 64-bit access by the assertion of **ACK64#** in response to a 64-bit request.

The Expansion ROM unit uses the ATU inbound transaction queue and the inbound read data queue.

When operating in the conventional PCI mode, the address of the inbound delayed read cycle is entered into the ITQ queue and the delayed read completion data is returned in the IRQ. That is, inbound reads to the Expansion ROM window are handled as delayed transactions on the PCI bus.

When operating in the PCI-X mode, the address of the inbound read cycle is entered into the ITQ queue and the read completion data is returned in the IRQ. That is, inbound reads to the Expansion ROM window are handled as split transactions on the PCI bus. The internal bus initiator interface fills the IRQ read queue with the full byte count before generating the split completion transaction on the PCI bus. That is, the ATU generates a Read request on the internal bus with byte count set to the byte count specified in the PCI read. The PBI (Flash Interface) returns data in:

- either one or more 1024 byte split completion transactions when byte count is greater than or equal to 1024 bytes or
- one split completion with the full byte count when byte count is less than 1024 bytes.

Expansion ROM writes are not supported and result in a Target Abort.



2.6 ATU Queue Architecture

ATU operation and performance depends on the queueing mechanism implemented between the internal bus interface and PCI bus interface. As indicated in [Figure 4](#), the ATU queue architecture consists of separate inbound and outbound queues. The function of each queue is described in the following sections.

2.6.1 Inbound Queues

The inbound data queues of the ATU support transactions initiated on a PCI bus and targeted at either 81341 and 81342 local memory or a 81341 and 81342 memory mapped register. [Table 7](#) details the name and sizes of the ATU inbound data queues.

Table 7. Inbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
IWQ	Inbound Write Data Queue	4 KBytes (4*1KB)
IWADQ	Inbound Write Address Queue	4 Transaction Addresses
IRQ	Inbound Read Data Queue	4 KBytes (4*1KB)
IDWQ	Inbound Delayed Write address/data Queue	1 Transaction
ITQ	Inbound Transaction Queue	8 Addresses/Commands

2.6.1.1 Inbound Write Queue Structure

The ATU Inbound Write Queues consist of the inbound write data queue and the inbound write address queue. The inbound write data queue holds the data for memory write transactions moving from a PCI Bus to the internal bus and the address queue holds the corresponding address of the transactions in the data queues. The inbound write queue, IWQ, has a queue depth of 4 KBytes and moves write transactions from the PCI bus to the internal bus. The corresponding address queue, IWADQ, is capable of holding 4 address entries. The queue pair is capable of holding up to 4 memory write (or MWI when operating in the conventional PCI mode) transactions.

The following rules apply to the PCI bus interface and govern the acceptance of data into IWQ and address into the tail of the IWADQ:

- A memory write operation claimed by the target PCI interface on the PCI bus is accepted into the address and data queues when the IWADQ has at least one address entry available.
- When operating in the conventional PCI mode, when the IWQ reaches a full state while filling, a disconnect with data is signaled to the master of the transaction.
- When operating in the PCI-X mode, when the IWQ reaches a full state while filling, a disconnect at next ADB is signaled to the master of that transaction.

Memory write transactions are drained from the head of the queue when the master interface has acquired bus ownership and transaction ordering and priority have been satisfied (see [Section 2.6.3](#)). A memory write transaction is considered drained from the queue when the entire amount of data entered on the PCI bus has been accepted by the internal bus target. Error conditions resulting in the cancellation of a write transaction only flush the transaction at the head of the data and address queue. All other transactions within the queues are considered still valid.



2.6.1.2 Inbound Read Queue Structure

The inbound read queues are responsible for retrieving data from local memory and returning it to the PCI bus in response to a read transaction initiated from a PCI master. When operating in the conventional PCI mode, reads are handled as delayed transactions. When operating in the PCI-X mode reads are handled as split transactions. The address of the transactions are held in the ITQ. Up to 8 read requests can be stored in the ITQ. The read data is returned through IRQ.

When operating in the conventional PCI mode, the IRQ holds data from four PCI bus read transactions. The read request cycle on PCI latches the read command and the address into the ITQ when the cycle is first initiated by the PCI master. The ATU internal bus initiator interface takes the translated address and the command and performs a read on the internal bus. Reads can be any of the PCI memory read command types using the ATU inbound translation or an inbound configuration read using the specific configuration cycle translation. The data from the read on the internal bus is stored in the IRQ until the PCI master initiates a read cycle that matches the initial request cycle in both command and address. Any data left in an IRQ after the delivery of a completion cycle on PCI is flushed. This is possible since all internal bus memory is considered prefetchable with no read side effects.

When operating in the PCI-X mode, the IRQ may hold data from up to four PCI bus read transactions. The read request cycle on PCI latches the read command and the address into the ITQ when the cycle is first initiated by the PCI master. The ATU internal bus initiator interface takes the translated address and the command and performs a read on the internal bus. Reads can be any of the PCI memory read command types using the ATU inbound translation or an inbound configuration read using the specific configuration cycle translation. Once read data is available in the IRQ, the ATU generates one or more split completions to return read data to the PCI requester.

When operating in the conventional PCI mode, the exact amount of data (byte count) read by the master state machine on the internal bus interface depends upon the read command used and how much data the Internal Bus target device delivers. [Table 8](#) shows the amount of data attempted to be read for the different memory read commands for the ATU, when operating in the conventional PCI mode.

Internal bus error conditions override all prefetch amounts (i.e., a master-abort and target-abort conditions).

Table 8. Inbound Read Prefetch Data Sizes

PCI Read Command	Prefetch Size (Bytes)
Memory Read	4 to 32
Memory Read Line	4 to 128
Memory Read Multiple	4 to 1024



2.6.1.3 Inbound Delayed Write Queue

The IDWQ is used specifically for inbound configuration write cycles to the ATU. I/O Write transactions are not accepted by the ATU and result in a Master Abort.

The IDWQ contains both the address and data of a configuration write cycle. When operating in the conventional PCI mode, the configuration writes are handled as delayed writes. When operating in the PCI-X mode, the configuration writes are handled as split writes. When the write cycle is initiated on the PCI bus, the address and data are entered into the 8 byte queue, and forwarded to the internal bus. The transaction is forwarded to the internal bus once transaction ordering has been satisfied. The status of the transaction (normal completion) is maintained in the IDWQ for return to the PCI master on the initiating bus. When operating in the PCI-X mode, a write completion message is generated by the ATU to indicate the successful execution of the configuration write transaction.

The IDWQ can only hold 32-bits of data and should never be accessed with **REQ64#** asserted per the *PCI Local Bus Specification*, Revision 2.3 which states that “only memory transactions support 64-bit data transfers”. In addition, the cycle should always return only 32-bits of data on the internal bus.

2.6.1.4 Inbound Transaction Queues Command Translation Summary

Table 9. PCI to Internal Bus Command Translation for All Inbound Transactions

PCI Command	Conventional Mode	PCI-X Mode	Internal Bus Command
Memory Write	a	a	Write
Memory Write and Invalidate	a		Write
Memory Write Block		a	Write
Alias to Memory Write Block		a	Write
Memory Read	a		Read
Memory Read Line	a		Read
Memory Read Multiple	a		Read
Memory Read Block		a	Read
Alias to Memory Read Block		a	Read
Memory Read DWORD		a	Read
Configuration Read	a	a	Read
Configuration Write	a	a	Write
Split Read Completion		a	Data Transfer
Split Write Completion		a	None
Split Completion Error Message		a	Data Transfer Abort
All Other Commands Not Claimed by the ATU	a	a	N/A



2.6.2 Outbound Queues

The outbound queues of the ATU are used to hold read and write transactions from the core processor directed at the PCI bus. Each ATU outbound queue structure has a separate read queue, write queue, and address queue. [Table 10](#) contains information about ATU outbound queues.

Table 10. Outbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
OWQ	Outbound Write Data Queue	4 KBytes (4*1024B)
OWADQ	Outbound Write Address Queue	4 Transaction Addresses
ORQ	Outbound Read Data Queue	2 or 4 KBytes (4* 512B or 4*1024B) ^a
OTQ	Outbound Transaction Queue	8 Addresses/Commands

a. The ORQ can be throttled between 2 Kbytes or 4 Kbytes depending on the setting of the Maximum Memory Read Byte Count (MMRBC) field of the PCI-X Command register (see [Section 2.13.55, “PCI-X Command Register - PCIXCMD” on page 196](#)). When the MMRBC is set to 512 bytes (default value), the ORQ is only capable of handling 2 Kbytes of SRC data, otherwise, the ORQ handles 4 Kbytes of SRC data.

The outbound queues are capable of holding outbound memory read, memory write, I/O read, and I/O write transactions. The type of transaction used is defined by the internal bus address and the command used on the internal bus. See [Section 2.2.2](#) and [Section 2.2.3](#) for details on outbound address translation.

When an internal bus agent initiates an outbound write transaction, the address is entered into the OWADQ (when not full). The data from the internal bus write is then entered into the OWQ and the transaction is forwarded to the PCI bus. When the write completes (or an error occurs), the address is flushed from the OWADQ. Data is flushed only for the master abort or target abort cases.

For outbound reads, the address is entered into the OTQ (when not full) and a split response termination is signaled to the requester on the internal bus. Read data is fetched and returned to the requester on the internal bus.

2.6.2.1 Relaxed Ordering and No Snoop Outbound Request Attributes

In PCI-X mode, the ATU may set the Relaxed Ordering (RO) bit 29 of the Requester Attributes and/or the No Snoop (NS) bit 30 of the Requester Attributes for an outbound request.

For outbound Memory Read and Memory write requests initiated by the ADMA the value of these attribute bits are controlled by the ADMA Descriptor Control Word (see [Section 5.3, “ADMA Descriptors” on page 466](#)).

For any other outbound requests, the NS and RO attribute bits are set to 0.

Note: The *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 permits the ATU to set the RO bit to '1' in the Requester Attributes only when enabled by bit 1 of the [“PCI-X Command Register - PCIXCMD” on page 196](#).



2.6.3 Transaction Ordering

Because the ATU can process multiple transactions, they must maintain proper ordering to avoid deadlock conditions and improve throughput. The ATU transaction ordering rules used by the 81341 and 81342 are listed in [Table 11](#) for the inbound direction and [Table 12 on page 93](#) for the outbound direction. The tables are based on the direction the transaction is moving, i.e. the data for outbound delayed read moves in the same direction as the data for an inbound write or the address/command for an inbound read. When operating in the PCI-X mode, the ATU ignores the Relaxed Ordering Attribute.

Note:

Outbound Non-Posted Writes are the result of Internal Bus Memory writes that are claimed by either the I/O translation window or the [Outbound Configuration Cycle Data Register - OCCDR](#). Though these write requests arrive on the PCI bus as non-posted write requests, it is important to note that from the Intel XScale® processor point of view, these internal bus memory write requests are posted into the Outbound ATU transaction queue. Furthermore, in PCI-X mode non-posted write requests have the potential to be split. Thus, even though a split write completion may be returned to the ATU on the PCI bus for a given outbound non-posted write request, the split write completion does not passed back through to the internal bus. Additionally, strong ordering between outbound memory (posted) write requests and outbound non-posted write requests are **not** maintained as indicated in [Table 12 on page 93](#).

For best performance, the user should designate the two Outbound Memory Windows as non-cacheable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 12, "ATU Outbound Data Flow Ordering Rules" on page 93](#). In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cacheable/non-bufferable and enforce the requirement in software.

Table 11. ATU Inbound Data Flow Ordering Rules

Row Pass Column? ^a	ATU Inbound Writes	Inbound Delayed Read Request (PCI mode)	Inbound Split Read Request (PCI-X mode)	Inbound Configuration Write Request	Outbound Split Read Completion
ATU Inbound Writes	No	Yes	Yes	Yes	Yes
Inbound Delayed Read Request (PCI mode)	No	No	NA	No	Yes
Inbound Split Read Request (PCI-X mode)	No	NA	No	No	Yes
Inbound Configuration Write Request	No	No	No	NA	Yes
Outbound Split Read Completion	No	Yes	Yes	Yes	Yes

a. Outbound Non-Posted Write Completions are not included in this table since these transactions are never passed back to the Internal Bus Requester (Intel XScale® processor). The reason is that from the Intel XScale® processor's point of view, these write requests are posted into the Outbound ATU transaction queue.

**Table 12. ATU Outbound Data Flow Ordering Rules**

Row Pass Column?	Outbound Write		Outbound Read Request	Inbound Read Completion		Inbound Write Completion	
	Posted	Non-Posted		DRC	SRC	DWC ^a	SWC ^b
Outbound Posted Write Request	No	Yes	Yes	Yes	Yes	Yes	Yes
Outbound Non-posted ^c Write Request	No	No	No	Yes	Yes	Yes	Yes
Outbound Read Request	No	No	No	Yes	Yes	Yes	Yes
Inbound Delayed Read Completion (DRC)	No	Yes	Yes	Yes	NA	Yes	NA
Inbound Split Read Completion (SRC)	No	Yes	Yes	NA	Yes	NA	No
Inbound Delayed Write Completion (DWC)	No	Yes	Yes	Yes	NA	NA	NA
Inbound Split Write Completion (SWC)	No	Yes	Yes	NA	Yes	NA	NA

- a. Since the Inbound DWR transaction queue is one-deep, the passing rule associated with DWC vs. DWC is moot (i.e., NA).
- b. Since the Inbound SWR transaction queue is one-deep, the passing rule associated with SWC vs. SWC is moot (i.e., NA).
- c. Outbound Non-posted writes include I/O writes and Configuration writes.

Definitions of the terms used in [Table 11](#) and [Table 12](#) are as follows. PCI terms are noted in parenthesis:

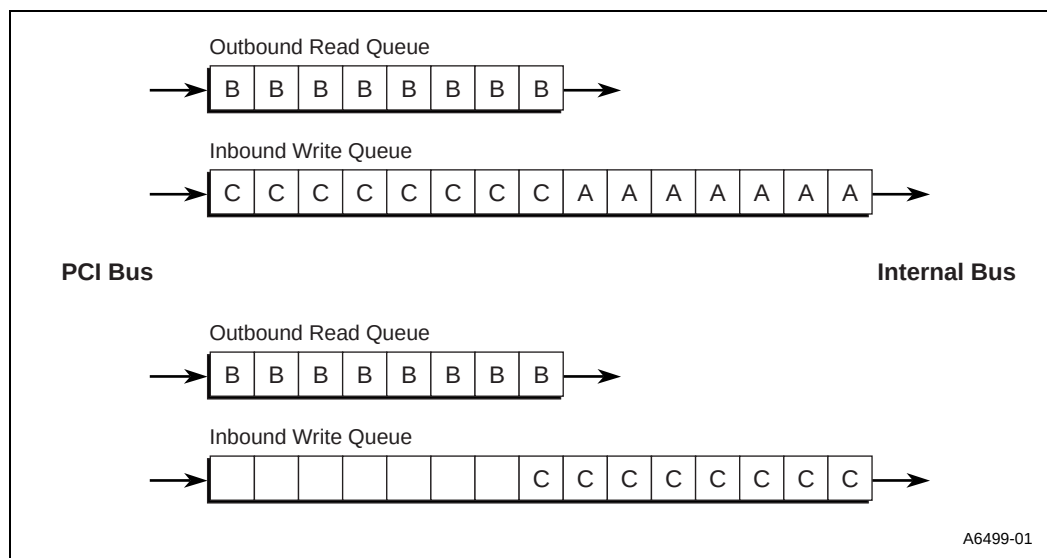
- Inbound Write (PMW) - Data from a write cycle initiated on PCI and targeted at the internal bus. Note that the address is in a separate transaction queue and is not referenced.
- Inbound Read Request (DRR-PCI mode and SRR-PCI-X mode) - address information from a read transactions retried or split on the PCI bus. Mastered on the internal bus to retrieve data for the Inbound Read Completion.
- Inbound Configuration Write Request - (DWR- PCI mode and SWR - PCI-X mode) - The address and data associated with a configuration write transaction from PCI and targeted at the ATU PCI configuration address space. Once completed on the internal bus, creates an Inbound Configuration Write Completion.
- Outbound Read Completion (SRC) - The data read on PCI in the process of being returned to the internal bus. This data is the completion cycle that results from an Outbound Read Request.
- Outbound Write (PMW) - The address and data from a write initiated on the internal bus and eventually completing on the PCI bus.
- Outbound Read Request (SRR) - The address/command of a split read cycle initiated on the internal bus. The read data is returned in the Outbound Read Completion cycle.
- Inbound Read Completion (DRC-PCI mode and SRC-PCI-X mode) - The data read on the internal bus in the process of being returned to the PCI bus. This data is the completion cycle for an Inbound Read Request.
- Inbound Configuration Write Completion (DWC-PCI mode and SWC-PCI-X mode) - The status of an inbound write configuration cycle traveling from the internal bus back towards the PCI bus.



These transaction ordering rules define the way data moves in both directions through the ATU. In [Table 11](#) and [Table 12](#) a **NO** response in a box means that based on ordering rules, the current transaction (the row) can not pass the previous transaction (the column) under any circumstance. A **Yes** response in the box means that the current transaction is *allowed* to pass the previous transaction but is not required to, based on whether a consistent view of data or prevention of deadlocks is needed.

In the case of inbound write operations, multiple transactions may exist within the IWQ and the corresponding IWADQ at any point in time. The ordering of these transactions is based on a time stamp basis. Transactions entering the queue are stamped with a relative time in relation to all other transactions moving in a similar direction.

Example 1. Inbound Queue Completion



In [Example 1 on page 94](#), the inbound write and outbound read queues of the ATU are shown. In this example, transaction A entered the write queue at **Time 0**. Next, the ATU entered read data into the outbound read queue at **Time 1** (Transaction B). Finally, before the previous transactions could be cleared, another inbound write, Transaction C, was entered into the IWQ. The ordering in [Table 11](#) states that nothing can pass an inbound write and therefore Transaction A must complete on the internal bus before Transaction B since an outbound read completion can not pass an inbound write. Also, Transaction A must complete before Transaction C since an inbound write can not pass another inbound write. Once Transaction A completes, Transaction C moves to the head of the IWQ. The two transactions at the head of the queues moving data in an inbound direction are now Transaction C, an inbound write, and Transaction B, an outbound read completion. Ordering states that an inbound write may pass an outbound read completion. This means that the arbitration mechanism now takes over to decide which completes. Note that ordering enforced the completion of Transaction A but arbitration dictated the completion of Transactions B and C.

The first action performed to determine which transaction is allowed to proceed (either inbound or outbound) is to apply the rules of ordering as defined in [Table 11](#) and [Table 12](#). Any box marked **No** must be satisfied first. For example, when an inbound read request is in ITQ and it was latched *after* the data in the IDWQ arrived (this is a configuration write), then ordering states that an Inbound Read Request may not pass an Inbound Configuration Write Request. Therefore, the Inbound Configuration Write Request must be cleared out of IDWQ before the Inbound Read Request is attempted on the internal bus. Once transaction ordering is satisfied, the boxes marked **Yes** are now resolved.



2.6.3.1 Transaction Ordering Summary

Table 13 and Table 14, define transaction ordering in relation to token assignment of the priority mechanism (this is discussed in Section 2.6.3). These tables are read as follows:

1. As the transaction enters the head of the respective queue, the question in column 2 is asked.
2. When all the answers in column 3 for a given transaction type assigns a token to the transaction at the head of the queue, a token is assigned. Otherwise, no token is assigned signifying that transaction ordering must first be satisfied. Any transaction with a token may be initiated on the bus.

Table 13. Inbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Inbound Write in IWQ	Is there an Inbound Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Read Request in ITQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Configuration Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Configuration Write Request in IDWQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Read Completion in ORQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token



Table 14. Outbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Outbound Write in OWQ	Is there an Outbound Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Read Request in OTQ	Is there an Outbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Outbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Configuration Write Completion in IRQ	Is there an Outbound Posted Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token
Inbound Read Completion in IRQ	Is there an Outbound Posted Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token
	Is there a Configuration Write Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token



2.6.4 Byte Parity Checking and Generation

The internal bus interface of the ATU supports byte-wise parity protection on the internal bus. This includes A_PARITY[4:0] and D_PARITY[15:0] on the address bus (A[35:0]) and the data bus (D[127:0]) respectively.

For an outbound write request (or inbound read completion) the internal bus interface verifies the write request address parity and data parity on the data cycles.

For an inbound read completion, the internal bus interface verifies the read completion data parity on the data cycles.

For an inbound write request, the ATU computes and appends write request address parity and data parity data parity prior to delivering the request to the internal bus.

For an inbound read request, the internal bus interface computes and appends read request address parity prior to delivering the request to the internal bus.

Note: The ATU forwards data parity across to the other interface. When a data parity error occurs on the internal bus interface, the same data issues on the PCI bus with either bad parity or uncorrectable ECC error. For inbound transactions the bad parity or uncorrectable ECC errors results in the ATU marking that data bad by corrupting the parity on the internal bus.

2.6.4.1 Parity Generation

Data parity signals include byte enables in the calculation. Table 15 lists the data bits that are used for the parity calculation. The parity bits are calculated by bit XORing the data bits as shown in Table 15. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is calculated as follows:

Equation 5. $D\_PARITY0 = D[0] \text{ XOR } D[1] \text{ XOR } D[2] \text{ XOR } D[3] \text{ XOR } D[4] \text{ XOR } D[5] \text{ XOR } D[6] \text{ XOR } D[7] \text{ XOR } WBE[0]$

Table 15. Parity Generation

Address/Data Parity Bit	Address/Data Bus	Address/Data Parity Bit	Address/Data Bus
A_PARITY4	A[35:32]	D_PARITY9	D[79:72], WBE[9]
A_PARITY3	A[31:24]	D_PARITY8	D[71:64], WBE[8]
A_PARITY2	A[23:16]	D_PARITY7	D[63:56], WBE[7]
A_PARITY1	A[15:8]	D_PARITY6	D[55:48], WBE[6]
A_PARITY0	A[7:0]	D_PARITY5	D[47:40], WBE[5]
D_PARITY15	D[127:120], WBE[15]	D_PARITY4	D[39:32], WBE[4]
D_PARITY14	D[119:112], WBE[14]	D_PARITY3	D[31:24], WBE[3]
D_PARITY13	D[111:104], WBE[13]	D_PARITY2	D[23:16], WBE[2]
D_PARITY12	D[103:96], WBE[12]	D_PARITY1	D[15:8], WBE[1]
D_PARITY11	D[95:88], WBE[11]	D_PARITY0	D[7:0], WBE[0]
D_PARITY10	D[87:80], WBE[10]		



2.6.4.2 Parity Checking

On an outbound request, address parity is checked on the address bus A[35:0]. The parity bits are checked by first bit XORing the address bits shown in [Table 15](#) with the corresponding address parity bits, and then verifying when the result of each of the XORed operations is equal to zero. As an example, the parity calculation for the lowest order byte of the address bus A[7:0] is carried as follows:

Equation 6.
$$\text{PARITY\_RESULT} = \text{A\_PARITY0} \text{ XOR } \text{A}[0] \text{ XOR } \text{A}[1] \text{ XOR } \text{A}[2] \text{ XOR } \text{A}[3] \text{ XOR } \text{A}[4] \text{ XOR } \text{A}[5] \text{ XOR } \text{A}[6] \text{ XOR } \text{A}[7]$$

The parity logic uses the following algorithm. This algorithm logs the error when an error is detected.

```
check address parity
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

On an outbound write request, data parity is checked on the data bus D[127:0]. The parity bits are checked by first bit XORing the data bits shown in [Table 15](#) with the corresponding data parity bits, and then verifying when the result of each of the XORed operations is equal to zero. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is carried as follows:

Equation 7.
$$\text{PARITY\_RESULT} = \text{D\_PARITY0} \text{ XOR } \text{D}[0] \text{ XOR } \text{D}[1] \text{ XOR } \text{D}[2] \text{ XOR } \text{D}[3] \text{ XOR } \text{D}[4] \text{ XOR } \text{D}[5] \text{ XOR } \text{D}[6] \text{ XOR } \text{D}[7] \text{ XOR } \text{WBE}[0]$$

A non-zero result from the above operation indicates a parity error.

The parity logic uses the following algorithm, and this algorithm logs the error when an error is detected.

```
check data parity
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

2.6.4.3 Parity Disabled

When software disables parity, the ATU does generate the parity byte for read completions, but does not check the address and data byte parity.



2.7 ATU Error Conditions

PCI and internal bus error conditions cause ATU state machines to exit normal operation and return to idle states. In addition, status bits are set to inform error handling code of exact cause of error condition. Error conditions and status can be found in the ATUSR. The basic flow for a PCI error is as follows:

- Set the bit in the ATU Status Register which corresponds to the error condition (master abort, target abort, etc.)
- Set the bit in the ATU Interrupt Status Register which corresponds to the error condition (master abort, target abort, etc.). This function is maskable for all PCI error conditions.
- The setting of the bit in the ATU Interrupt Status Register results in an interrupt being driven to the Intel XScale® processor.

Error conditions on one side of the ATU are generally propagated to the other side of the ATU and have different effects depending on the error. Error conditions and their effects are described in the following sections.

PCI bus error conditions and the action taken on the bus are defined within the *PCI Local Bus Specification*, Revision 2.3, and the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0. The ATU adheres to the error conditions defined within the PCI specification for both requester and target operation. Error conditions on the internal bus are caused by an ECC error from the Memory Controller, (see [Section 7.5, "ECC Interrupts/Error Conditions" on page 621](#) for details on memory controller error conditions), an Internal Bus Byte Parity Error, or by incorrect addressing resulting in an internal master abort. All actions on the PCI Bus for error situations are dependent on the error control bits found in the ATU Command Register (see [Section 2.13.5, "ATU Command Register - ATUCMD" on page 153](#)) for both Conventional and PCI-X modes. For PCI-X mode, the error response is also dependent on an error control bit in the PCI-X Command Register (see [Section 2.13.55, "PCI-X Command Register - PCIXCMD" on page 196](#)). In addition, for PCI-X Mode 2 only, the error response also depends on the ECC Control and Status Register (see [Section 2.13.57, "ECC Control and Status Register - ECCCSR" on page 200](#)).

The 81341 and 81342 operates in parity mode (when enabled) for conventional PCI (PCI-33, PCI-66), and PCI-X Mode 1 (PCI-X 66, PCI-X 133). For PCI-X Mode 2, the 81341 and 81342 functions in ECC mode. Parity errors, single-bit ECC errors (when correction is disabled), and multi-bit ECC errors are uncorrectable errors. In ECC mode, all single-bit errors are corrected when error correction is enabled and the transaction completes.

The following sections detail all ATU error conditions on the PCI and 81341 and 81342 internal bus, action taken on these conditions, and status and control bits associated with error handling.



2.7.1 Uncorrectable Address and Uncorrectable Attribute Errors on the PCI Interface

The ATUs must detect and report uncorrectable address and attribute (PCI-X mode only) errors for transactions on the PCI bus. When an uncorrectable address or attribute error occurs on the PCI interface of the ATU, the 81341 and 81342 performs the following actions based on the constraints specified:

- In Conventional mode, when the Parity Error Response bit in ATUCMD is set, the ATU ignores (Master-Abort) the transaction by not asserting **DEVSEL#**. When clear, the transaction proceeds normally.
- In PCI-X mode, when the Parity Error Response bit in ATUCMD is set, the ATU completes the transaction on the PCI bus as when no error had occurred, but the request or completion is not forwarded to the internal bus. When clear, the transaction proceeds normally.
- Assert **SERR#** when the **SERR#** Enable bit and the Parity Error Response bit in the ATUCMD are set. When the ATU asserts **SERR#**, additional actions is taken:
 - Set the **SERR#** Asserted bit in the ATUSR
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR (see [Section 2.13.40, "ATU Configuration Register - ATUCR" on page 182](#)) is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- Set the Detected Uncorrectable Address or Attribute Error bit in PCSR (PCI Configuration and Status Register).
- Set the Detected Parity Error bit in the ATUSR. When the ATU sets the Detected Parity Error bit, additional actions is taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the "ECC Control and Status Register - ECCCSR" on [page 200](#), the "ECC First Address Register - ECCFAR" on [page 203](#), the "ECC Second Address Register - ECCSAR" on [page 204](#), and the "ECC Attribute Register - ECCAR" on [page 205](#) for the transaction

Note: The Detected Parity Error bit with its' associated interrupt along with the Detected Uncorrectable Address or Attribute Error bit provides software with the ability to distinguish between an Uncorrectable Address or Attribute error versus an Uncorrectable Data Error during a Detected Parity error interrupt.



2.7.2 Correctable Address and Correctable Attribute Errors on the PCI Interface

In PCI-X Mode 2 (when single-bit correction is enabled), the ATUs must detect and report correctable address and attribute (PCI-X mode only) errors for transactions on the PCI bus. When a correctable address or attribute error occurs on the PCI interface of the 81341 and 81342, the ATU performs the following actions based on the constraints specified:

- The error is corrected and the ATU completes the transaction on the PCI bus as when no error had occurred. Then, the transaction is forwarded to the internal bus normally.
- Update the ["ECC Control and Status Register - ECCCSR" on page 200](#), the ["ECC First Address Register - ECCFAR" on page 203](#), the ["ECC Second Address Register - ECCSAR" on page 204](#), and the ["ECC Attribute Register - ECCAR" on page 205](#) for the transaction.
 - When the ATU Detected Correctable Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Correctable Error bit in the ATUISR. When set, no action.

Note: The ECCCSR provides information on the transaction phase in which the correctable error occurred.



2.7.3 Uncorrectable Data Errors on the PCI Interface

Two kinds of uncorrectable data errors can occur on the PCI interface: errors as an initiator and errors as a target.

Errors encountered as an initiator:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completions
- Inbound Configuration Write Completion Messages

As an initiator, the ATU provides an error response for uncorrectable data errors on outbound reads, and uncorrectable data errors occurring at the target for outbound writes. However, there is no error response for uncorrectable data errors on inbound configuration write completion messages and inbound read completions.

Errors encountered as a target:

- Inbound Read Request (Immediate Data Transfer)
- Inbound Write Request
- Outbound Read Completions
- Outbound Split (I/O or Configuration) Uncorrectable Write Data Error Messages
- Inbound Configuration Write
- Split Completion Messages

As a target, the ATU provides an error response for uncorrectable data errors on inbound writes, outbound read completions, outbound uncorrectable split write data error messages, inbound configuration writes, and split completion messages. However, there is no error response for uncorrectable data errors on inbound reads.



2.7.3.1 Outbound Read Request Uncorrectable Data Errors

2.7.3.1.1 Immediate Data Transfer

As an initiator, the ATU may encounter this error condition in Conventional or PCI-X mode when the target transfers data immediately rather than signalling a Retry⁵ (Conventional Delayed Read Request) or a Split Response Termination (PCI-X Split Read Request).

Uncorrectable data errors occurring during read operations initiated by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the data phase in which the uncorrectable data error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions are taken:
 - The Master Parity Error bit in the ATUSR is set.
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the ATU is operating in the PCI-X mode, the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
 - The read completion are aborted on the internal bus of the 81341 and 81342.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the “ECC Control and Status Register - ECCCSR” on page 200, the “ECC First Address Register - ECCFAR” on page 203, the “ECC Second Address Register - ECCSAR” on page 204, and the “ECC Attribute Register - ECCAR” on page 205 for the transaction.

5. Retry terminations may also be signaled in PCI-X mode when the target is too busy to handle the current request. However, this is not the same as a Delayed Read Request in Conventional PCI mode since the requester is not required or expected by the target to return with the same read request.



2.7.3.1.2 Split Response Termination

As an initiator, the ATU may encounter this error condition in PCI-X mode when the target signals a Split Response Termination.

Parity errors occurring during Split Response Terminations of Read Requests by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the Split Response Termination in which the parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions are taken:
 - The Master Parity Error bit in the ATUSR is set.
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the “[ECC Control and Status Register - ECCCSR](#)” on [page 200](#), the “[ECC First Address Register - ECCFAR](#)” on [page 203](#), the “[ECC Second Address Register - ECCSAR](#)” on [page 204](#), and the “[ECC Attribute Register - ECCAR](#)” on [page 205](#) for the transaction.

The Outbound Read Request remains enqueued in the ATU since the completer is initiating completion transactions that are associated with this request.



2.7.3.2 Outbound Write Request Uncorrectable Data Errors

2.7.3.2.1 Outbound Writes that are not MSI (Message Signaled Interrupts)

As an initiator, the ATU may encounter this error condition when operating in either the Conventional or PCI-X modes.

Uncorrectable Data Errors occurring during write operations initiated by the ATU may record the assertion of **PERR#** from the target on the PCI Bus. In PCI-X mode, this includes the assertion of **PERR#** from the target on the PCI Bus following the split response termination of a non-posted write request. When an error occurs, the ATUs continue writing data to the target to clear the OWQ of the current outbound write transaction. Specifically, the following actions with the given constraints are taken by the ATU:

- When **PERR#** is sampled active and the Parity Error Response bit in the ATUCMD is set, set the Master Parity Error bit in the ATUSR. When the Parity Error Response bit in the ATUCMD is clear, no action is taken. When the Master Parity Error bit in the ATUSR is set, additional actions are taken:
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the ATU is operating in the PCI-X mode, the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action

Outbound uncorrectable write data errors, do not result in a master completion. In addition, when the target terminates the transaction (disconnect), the ATU master must reinitiate the transaction to clear the data from the OWQ.

2.7.3.2.2 MSI Outbound Writes

As an initiator, the ATU may encounter this error condition when operating in either the Conventional or PCI-X modes.

Uncorrectable Data Errors occurring during MSI write operations initiated by the ATU may record the assertion of **PERR#** from the target on the PCI Bus. When an error occurs, the ATU completes the transaction normally. Then, the following actions with the given constraints are taken by the ATU:

- When **PERR#** is sampled active and the Parity Error Response bit in the ATUCMD is set, set the Master Parity Error bit in the ATUSR. When the Parity Error Response bit in the ATUCMD is clear, no action is taken. When the Master Parity Error bit in the ATUSR is set, additional actions are taken:
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.



2.7.3.3 Inbound Read Completions Uncorrectable Data Errors

As an initiator, ATU may encounter this error condition when operating in PCI-X mode.

When as the completer of a Split Read Request the ATU observes **PERR#** assertion during the split completion transaction, the ATU attempts to complete the transaction normally and no further action are taken.

2.7.3.4 Inbound Configuration Write Completion Message Uncorrectable Data Errors

As an initiator, ATU may encounter this error condition when operating in PCI-X mode.

When as the completer of a Configuration (Split) Write Request the ATU observes **PERR#** assertion during the split completion transaction, the ATU attempts to complete the transaction normally and no further action are taken.

2.7.3.5 Inbound Read Request Uncorrectable Data Errors

2.7.3.5.1 Immediate Data Transfer

As a target, the ATU may encounter this error when operating in the Conventional PCI or PCI-X modes.

Inbound read uncorrectable data errors occur when read data delivered from the IRQ is detected as having bad parity by the initiator of the transaction who is receiving the data. The initiator may optionally report the error to the system by asserting **PERR#**. As a target device in this scenario, no action is required and no error bits are set.

2.7.3.5.2 Split Response Termination

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Inbound read uncorrectable data errors occur during the Split Response Termination. The initiator may optionally report the error to the system by asserting **PERR#**. As a target device in this scenario, no action is required and no error bits are set.

2.7.3.6 Inbound Write Request Uncorrectable Data Errors

As a target, ATU may encounter this error when operating in Conventional or PCI-X modes.

Uncorrectable Data errors occurring during write operations received by the ATU may assert **PERR#** on the PCI Bus. When an error occurs, the ATU continues accepting data until the initiator of the write transaction completes or a queue fill condition is reached. Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in PCI-X mode) following the data phase in which uncorrectable data error is detected on the bus. This is only done when the Parity Error Response bit in ATUCMD is set.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the "ECC Control and Status Register - ECCCSR" on page 200, the "ECC First Address Register - ECCFAR" on page 203, the "ECC Second Address Register - ECCSAR" on page 204, and the "ECC Attribute Register - ECCAR" on page 205 for the transaction.



2.7.3.7 Outbound Read Completion Uncorrectable Data Errors

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Uncorrectable Data errors occurring during read completion transactions that are claimed by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the data phase in which the uncorrectable data error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions are taken:
 - The Master Parity Error bit in the ATUSR is set.
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
 - The read completion is aborted on the internal bus of the 81341 and 81342.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the [“ECC Control and Status Register - ECCCSR” on page 200](#), the [“ECC First Address Register - ECCFAR” on page 203](#), the [“ECC Second Address Register - ECCSAR” on page 204](#), and the [“ECC Attribute Register - ECCAR” on page 205](#) for the transaction.



2.7.3.8 Outbound Split Write Uncorrectable Data Error Message

The ATU claims a Split Completion Error Message that indicates an uncorrectable data error has occurred on one of the ATUs non-posted (I/O or Configuration) write requests (Message Class = 2h, Message Index = 01h -- Uncorrectable Split Write Data Error or Message Class = 1h, Message Index = 02h --Uncorrectable Write Data Error).

When the ATU receives a Split Completion Error Message indicating an uncorrectable data error for an outstanding Configuration or I/O (split) write request, the error is recorded, and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- When the Parity Error Response bit in the ATUCMD is set, these actions are taken:
 - The Master Parity Error bit in the ATUSR is set.
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- The Received Split Completion Error Message bit in the PCIXSR is set (based on bit 30 of the completer attributes being set). When the ATU sets this bit, additional actions are taken:
 - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- The transaction associated with the Split Completion Error Message is discarded.



2.7.3.9 Inbound Configuration Write Request

As a target, the ATU may encounter this error when operating in the Conventional or PCI-X modes.

2.7.3.9.1 Conventional PCI Mode

To allow for correct data parity calculations for delayed write transactions, the ATU delays the assertion of **STOP#** (signalling a Retry) until **PAR** is driven by the master. A parity error during a delayed write transaction (inbound configuration write cycle) can occur in any of the following parts of the transactions:

- During the initial Delayed Write Request cycle on the PCI bus when the ATU latches the address/command and data for delayed delivery to the internal configuration register.
- During the Delayed Write Completion cycle on the PCI bus when the ATU delivers the status of the operation back to the original master.

The 81341 and 81342 ATU PCI interface has the following responses to a delayed write parity error for inbound transactions during Delayed Write Request cycles with the given constraints:

- When the Parity Error Response bit in the ATUCMD is set, the ATU asserts **TRDY#** (disconnects with data) and two clock cycles later asserts **PERR#** notifying the initiator of the parity error. The delayed write cycle is not enqueued and forwarded to the internal bus.
- When the Parity Error Response bit in the ATUCMD is cleared, the ATU retries the transaction by asserting **STOP#** and enqueues the Delayed Write Request cycle to be forwarded to the internal bus. **PERR#** is not asserted.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the [“ECC Control and Status Register - ECCCSR” on page 200](#), the [“ECC First Address Register - ECCFAR” on page 203](#), the [“ECC Second Address Register - ECCSAR” on page 204](#), and the [“ECC Attribute Register - ECCAR” on page 205](#) for the transaction.

For the original write transaction to be completed, the initiator retries the transaction on the PCI bus and the ATU returns the status from the internal bus, completing the transaction.

For the Delayed Write Completion transaction on the PCI bus where a data parity error occurs and therefore does not agree with the status being returned from the internal bus (i.e. status being returned is normal completion) the ATU performs the following actions with the given constraints:

- When the Parity Error Response Bit is set in the ATUCMD, the ATU asserts **TRDY#** (disconnects with data) and two clocks later asserts **PERR#**. The Delayed Completion cycle in the IDWQ remains since the data of retried command did not match the data within the queue.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.



2.7.3.9.2 PCI-X Mode

Uncorrectable Data errors occurring during configuration write operations received by the ATU may cause **PERR#** assertion and delivery of a Split Completion Error Message on the PCI Bus. When an error occurs, the ATU accepts the write data and complete with a Split Response Termination. Specifically, the following actions with the given constraints are then taken by the ATU:

- When the Parity Error Response bit in the ATUCMD is set, **PERR#** is asserted three clocks cycles following the Split Response Termination in which the uncorrectable data error is detected on the bus. When the ATU asserts **PERR#**, additional actions are taken:
 - An Uncorrectable Split Write Data Error message (with message class=2h - completer error and message index=01h - Uncorrectable Split Write Data Error) is initiated by the ATU on the PCI bus that addresses the requester of the configuration write.
 - When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.
 - The Split Write Request is not enqueued and forwarded to the internal bus.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the ["ECC Control and Status Register - ECCCSR" on page 200](#), the ["ECC First Address Register - ECCFAR" on page 203](#), the ["ECC Second Address Register - ECCSAR" on page 204](#), and the ["ECC Attribute Register - ECCAR" on page 205](#) for the transaction.



2.7.3.10 Split Completion Messages

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Uncorrectable Data errors occurring during Split Completion Messages claimed by the ATU may assert **PERR#** (when enabled) or **SERR#** (when enabled) on the PCI Bus. When an error occurs, the ATU accepts the data and complete normally. Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted three clocks cycles following the data phase in which the uncorrectable data error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions are taken:
 - The Master Parity Error bit in the ATUSR is set.
 - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
 - When the **SERR#** Enable bit in the ATUCMD is set, and the Uncorrectable Data Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**; otherwise no action is taken. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- When the SCE bit (Split Completion Error -- bit 30 of the Completer Attributes) is set during the Attribute phase, the Received Split Completion Error Message bit in the PCIXSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- For PCI-X Mode 2, update the "ECC Control and Status Register - ECCCSR" on page 200, the "ECC First Address Register - ECCFAR" on page 203, the "ECC Second Address Register - ECCSAR" on page 204, and the "ECC Attribute Register - ECCAR" on page 205 for the transaction.
- The transaction associated with the Split Completion Message is discarded.
- When the discarded transaction was a read, the read completion is aborted on the internal bus of the 81341 and 81342.



2.7.4 Correctable Data Errors on the PCI Interface

When the 81341 and 81342 PCI interface is operating in Mode 2 and Single-Bit Correction is enabled, correctable data errors may occur on the PCI bus.

Two kinds of correctable data errors can occur on the PCI interface: errors as an initiator and errors as a target.

Errors encountered as an initiator:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completions
- Inbound Configuration Write Completion Messages

As an initiator, the ATU provides no error response for correctable data errors.

Errors encountered as a target:

- Inbound Read Request (Immediate Data Transfer)
- Inbound Write Request
- Outbound Read Completions
- Inbound Configuration Write
- Split Completion Messages

As a target, the ATU provides an error response for correctable data errors on inbound writes, outbound read completions, inbound configuration writes, and split completion messages. However, the ATU provides no error response for correctable data errors on inbound read requests.

In general, when the ATU is receiving data from the PCI bus as a target, any correctable data errors are corrected and logged. Otherwise, the ATU functions as when no error had occurred.

2.7.4.1 Inbound Read Request Correctable Data Errors

2.7.4.1.1 Immediate Data Transfer

As a target device in this scenario, no action is required and no error bits are set.

2.7.4.1.2 Split Response Termination

As a target device in this scenario, no action is required and no error bits are set.

2.7.4.2 Inbound Write Request Correctable Data Errors

As a target device, when an inbound write request correctable data error is detected, the following actions are taken:

- Error is corrected and ATU completes the transaction on the PCI bus as when no error had occurred. Then, the transaction is normally forwarded to the internal bus.
- Update “ECC Control and Status Register - ECCCSR” on page 200, “ECC First Address Register - ECCFAR” on page 203, “ECC Second Address Register - ECCSAR” on page 204, and “ECC Attribute Register - ECCAR” on page 205 for transaction.
 - When ATU Detected Correctable Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Correctable Error bit in the ATUISR. When set, no action.



2.7.4.3 Outbound Read Completion Correctable Data Errors

As a target device, when an outbound read completion correctable data error is detected, the following actions are taken:

- The error is corrected and the ATU completes the transaction on the PCI bus as when no error had occurred. Then, the transaction is forwarded to the internal bus normally.
- Update the “ECC Control and Status Register - ECCCSR” on page 200, the “ECC First Address Register - ECCFAR” on page 203, the “ECC Second Address Register - ECCSAR” on page 204, and the “ECC Attribute Register - ECCAR” on page 205 for the transaction.
 - When the ATU Detected Correctable Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Correctable Error bit in the ATUISR. When set, no action.

2.7.4.4 Inbound Configuration Write Request

As a target device, when an inbound configuration write request correctable data error is detected, the following actions are taken:

- The error is corrected and the ATU completes the transaction on the PCI bus as when no error had occurred. Then, the transaction is forwarded to the internal bus normally.
- Update the “ECC Control and Status Register - ECCCSR” on page 200, the “ECC First Address Register - ECCFAR” on page 203, the “ECC Second Address Register - ECCSAR” on page 204, and the “ECC Attribute Register - ECCAR” on page 205 for the transaction.
 - When the ATU Detected Correctable Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Correctable Error bit in the ATUISR. When set, no action.

2.7.4.5 Split Completion Messages

As a target device, when a split completion message correctable data error is detected, the following actions are taken:

- The error is corrected and the ATU completes the transaction on the PCI bus as when no error had occurred. Then, the transaction is forwarded to the internal bus normally.
- Update the “ECC Control and Status Register - ECCCSR” on page 200, the “ECC First Address Register - ECCFAR” on page 203, the “ECC Second Address Register - ECCSAR” on page 204, and the “ECC Attribute Register - ECCAR” on page 205 for the transaction.
 - When the ATU Detected Correctable Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Correctable Error bit in the ATUISR. When set, no action.

2.7.5 Master Aborts on the PCI Interface

As an initiator on the PCI bus, the ATU can encounter master abort conditions during:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completion
- Inbound Configuration Write Completion Message

As a target, the ATU PCI interface is capable of signaling a master abort case during:

- Uncorrectable Address Error (Conventional Mode)
- Inbound Read Request (PCI-X Mode)

2.7.5.1 Master Aborts for Outbound Read or Write Request

This error may be encountered in both the Conventional and the PCI-X modes. For an Outbound transaction, there are two ways in which a Master-Abort may be signaled to the ATU:

1. In the Conventional or PCI-X modes, a master abort is signaled when the target of the transaction does not assert **DEVSEL#** within 5 clocks (7 clocks when operating in the PCI-X mode) of the assertion of **FRAME#**.
2. In PCI-X mode, ATU may enqueue a Split request (Read or Write) on target-side interface of a PCI-to-PCI Bridge. When PCI-to-PCI Bridge detects a Master Abort on requester-side interface for that Split Request, master abort is signaled to ATU through a Master-Abort Split Completion Error Message (class=1h - bridge error and index=00h - Master Abort). The following actions with given constraints are performed by ATU when a master abort is detected by the PCI initiator interface or the PCI target interface receives a Master-Abort Split Completion error message:
 - Set the Master Abort bit (bit 13) in the ATUSR.
 - When the ATU PCI Master Abort Interrupt Mask bit in the ATUIMR is clear, set the PCI Master Abort bit in the ATUISR. When set, no action.
 - When an outbound write or inbound completion, flush data and address.
 - When the transaction is an MSI outbound write and the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action
 - When operating in PCI-X mode and Master-Abort is signaled via a Split Completion Error Message, the Received Split Completion Error Message bit in PCIXSR is set. When ATU sets this bit, additional actions are taken:
 - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
 - For an Outbound Read request, generate a split completion error message (class=1h - 81341 and 81342 Outbound Request error and index=00h - master abort) on the internal bus.
 - Flush the address from the OTQ.



2.7.5.2 Inbound Read Completion or Inbound Configuration Write Completion Message

The ATU encounters this error only in the PCI-X mode.

A master abort is signaled when the target of the transaction does not assert **DEVSEL#** within 7 clocks of the assertion of **FRAME#**.

When the ATU is signaled a Master-Abort while initiating either a Split Read Completion Transaction or a Split Write Completion Message, the ATU discards the Split Completion and take no further action.

2.7.5.3 Master-Aborts Signaled by the ATU as a Target

2.7.5.3.1 Uncorrectable Address Errors

The ATU can only signal this error during an Uncorrectable Address Error in the Conventional mode.

Please see [Section 2.7.1, "Uncorrectable Address and Uncorrectable Attribute Errors on the PCI Interface"](#) on page 100 for details on the ATU response to an Uncorrectable Address Error in the Conventional mode.

2.7.5.3.2 Internal Bus Master-Abort

The ATU can only signal this error during an internal bus master abort in the PCI-X mode.

Please see [Section 2.7.9.1, "Master Abort on the Internal Bus"](#) on page 120 for details on the ATU response to an Internal Bus Master Abort in the PCI-X mode.



2.7.6 Target Aborts on the PCI Interface

As an initiator on the PCI bus, the ATU can encounter target abort conditions during:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completion
- Inbound Configuration Write Completion Message

As a target, the ATU PCI interface is capable of signaling a target abort case during:

- Inbound Read Request (PCI-X and Conventional Modes)
- Inbound Write Request to EROM memory space (PCI-X and Conventional Modes)

2.7.6.1 Target Aborts for Outbound Read Request or Outbound Write Request

This error can be encountered by the ATU in both the Conventional and PCI-X modes. For an Outbound transaction, there are two ways in which a Target-Abort may be signaled to the ATU:

1. In the Conventional or PCI-X modes, a target abort is signaled when the target of the transaction simultaneously deasserts **DEVSEL#**, deasserts **TRDY#**, and asserts **STOP#**.
2. In PCI-X mode, ATU may enqueue a Split request (Read or Write) on target-side interface of a PCI-to-PCI Bridge. When PCI-to-PCI Bridge detects a Target Abort on requester-side interface for that Split Request, target abort is signaled to ATU through a Target-Abort Split Completion Error Message (class=1h - bridge error and index=01h - Target Abort). The following actions with the given constraints are performed by the ATU when a target abort is detected by the PCI initiator interface or the PCI target interface receives a Target-Abort Split Completion error message:
 - Set the Target Abort (master) bit (bit 12) in the ATUSR.
 - When the ATU PCI Target Abort (master) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (master) bit in the ATUISR. When set, no action.
 - When the transaction is an MSI outbound write and the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**; otherwise, no action is taken. When the ATU asserts **SERR#**, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR.
 - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
 - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
 - When operating in the PCI-X mode and the Target-Abort is signaled via a Split Completion Error Message, the Received Split Completion Error Message bit in the PCIXSR is set. When the ATU sets this bit, additional actions are taken:
 - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
 - For an Outbound Read request, the read completion is aborted on the internal bus.
 - Flush the address from the OTQ.



2.7.6.2 Inbound Read Completion or Inbound Configuration Write Completion Message

The ATU encounters this error only in the PCI-X mode.

A target abort is signaled when the target of the transaction simultaneously deasserts **DEVSEL#**, deasserts **TRDY#**, and asserts **STOP#**.

When the ATU is signaled a Target-Abort while initiating either a Split Read Completion Transaction or a Split Write Completion Message, the ATU discards the Split Completion and take no further action

2.7.6.3 Target-Aborts Signaled by the ATU as a Target

2.7.6.3.1 Internal Bus Master Abort

A target abort can be signaled by the ATU during an inbound read request where the internal bus cycle resulted in a master abort on the Internal Bus.

Please see [Section 2.7.9.1, "Master Abort on the Internal Bus" on page 120](#) for details on the ATU response to an Internal Bus Master Abort.

2.7.6.3.2 Internal Bus Target Abort

A target abort can be signaled by the ATU during an inbound read request where the internal bus cycle resulted in a Target Abort from the memory controller due to a non-recoverable multi-bit ECC error.

Please see [Section 2.7.9.2, "Target Abort on the Internal Bus" on page 122](#) for details on the ATU response to an Internal Bus Target Abort.

2.7.6.3.3 Inbound EROM Memory Write

Since the EROM memory window is defined to be read-only by the *PCI Local Bus Specification*, Revision 2.3, the ATU target-aborts when an inbound write transaction is claimed by the EROM memory window.

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI target interface during an inbound EROM Memory write transaction:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
 - When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.



2.7.7 Corrupted or Unexpected Split Completions

Warning: When any of the errors discussed in this section actually occur, a catastrophic system failure is likely to result from which the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 provides no recovery mechanism. In these cases, the ATU may be communicating with a non-compliant target device or the system may not be configured properly.

2.7.7.1 Completer Address

The ATU only asserts **DEVSEL#** for split completion transactions where the Sequence ID (Requester ID & Tag) matches that of a currently outstanding split request in the OTQ.

Conversely, the ATU does not assert **DEVSEL#** for any split completion transaction where either the Requester ID does not match that of the ATU or the Tag does not match that of any currently outstanding split request. No further action is taken.

When the Sequence ID of a split completion transaction matches that of an outstanding request, but the Lower Address field is not valid, the ATU accepts the split completion transaction in its' entirety according to the invalid Lower Address field and set the Unexpected Split Completion bit in the PCIXSR. No further action is taken.

2.7.7.2 Completer Attributes

When the Sequence ID of a split completion transaction matches that of an outstanding request, but the Byte Count is not valid, the ATU accepts the split completion transaction in its' entirety according to the invalid byte count field and set the Unexpected Split Completion bit in the PCIXSR. In this case, the ATU discards all the data. No further action is taken.



2.7.8 SERR# Assertion and Detection

The ATU is capable of reporting error conditions through the use of the **SERR#** output.

The following conditions may result in the assertion of **SERR#** by the ATU:

- An uncorrectable address error (or an uncorrectable attribute error when operating in the PCI-X mode) is detected by the ATU PCI interface (see [Section 2.7.1, "Uncorrectable Address and Uncorrectable Attribute Errors on the PCI Interface" on page 100](#) for details).
- A Master Data Parity Error is recorded in the ATUSR while operating in the PCI-X mode (see [Section 2.7.3, "Uncorrectable Data Errors on the PCI Interface" on page 102](#) for details).
- An outbound MSI write transaction is either signaled a Master-Abort or a Target-Abort by the target.
- An inbound write transaction is master aborted on the internal bus (see [Section 2.7.9.1, "Master Abort on the Internal Bus" on page 120](#) for details).
- The **SERR#** Manual Assertion bit in the ATUCR has been set and the **SERR#** Enable bit is set in the ATUCMD.

Note that the **SERR#** manual assertion bits must be cleared manually before they can be set again resulting in **SERR#** asserted. Refer to [Section 2.13.40, "ATU Configuration Register - ATUCR" on page 182](#) for details.

The following actions with the given constraints are performed by the ATU when **SERR#** is asserted by the PCI interface:

- Set the **SERR#** Asserted bit in the ATUSR.
- When the ATU **SERR#** Asserted Interrupt Mask bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
- When **SERR#** is asserted and the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action.

The following actions with the given constraints are performed by the ATU when **SERR#** is detected by the PCI interface:

- When **SERR#** is detected and the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action.

Note: Whenever the ATU asserts **SERR#**, both the asserted and detected status bits may be set in the corresponding ISR. To mask an interrupt to the core when the ATU asserts **SERR#**, the **SERR#** asserted mask bit must be set and the **SERR#** detected interrupt enable bit must be clear.



2.7.9 Internal Bus Error Conditions

An internal bus error results in a bit being set in the Interrupt Status Registers at which time an interrupt is driven to the Intel XScale® processor. Unlike PCI errors, internal bus error conditions are not maskable.

The following sections detail internal bus error conditions for the ATU.

2.7.9.1 Master Abort on the Internal Bus

A master abort on the internal bus is seen by the ATU when the inbound translated address presented on the internal bus is not claimed.

2.7.9.1.1 Inbound Write Request

The following action with the given constraints are performed by the ATU when a master abort is detected by the internal master interface during an inbound write request transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR.
- When the Inbound Error **SERR#** Enable bit is set in the ATUIMR and the **SERR#** Enable bit is set in the ATUCMD, assert **SERR#** on the PCI interface. When both bits are not set, take no action. When **SERR#** is asserted, additional actions are taken:
 - Set the **SERR#** Asserted bit in the ATUSR
 - When the ATU **SERR#** Asserted Interrupt Mask bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action
 - When the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action
- Flush the transaction that was master aborted from the IWQ.

The Internal Bus Master Abort bit is non-maskable and always results in an interrupt being driven to the core processor.



2.7.9.1.2 Inbound Read Request

When operating in the Conventional mode, the following actions with the given constraints are performed by the ATU when a master abort is detected by the internal initiator interface during an inbound read transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR
- Return a target abort condition to the initiating master during the delayed completion cycle on the PCI bus. No data is ever read from the internal bus and returned to the PCI bus.

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI interface during an inbound delayed read completion cycle:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
- When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.
- Flush the transaction that was master aborted from the ITQ after the target abort is delivered on the PCI interface.

When operating in the PCI-X mode, the following actions with the given constraints are performed by the ATU when a master abort is detected by the internal master interface during an inbound split read transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR.
- Generate a Split Completion Error Message (with message class=2h - completer error and message index=80h - 81341 and 81342 internal bus master abort) on the PCI bus.
- When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.
- Flush the transaction that was master aborted.

Note: This split completion error message includes a device specific message index. The error handler would need to have knowledge of the device specific error messages of the 81341 and 81342 in order to fully diagnose the problem.

The Internal Bus Master Abort bit is non-maskable and always results in an interrupt being driven to the core processor.



2.7.9.2 Target Abort on the Internal Bus

Target Aborts can be seen by the internal bus requester interface during inbound read operations to the memory controller. During inbound read operations, the memory controller is capable of signalling a target abort when a multi-bit, unrecoverable ECC error is encountered. This can occur during any read operation.

Note target aborts are signalled on a Qword basis. When either Dword of a Qword target aborts, both are considered to have target aborted.

The Memory Controller is responsible for creating an interrupt to the Intel XScale® processor for any multi-bit ECC errors.

2.7.9.2.1 Conventional Mode

When operating in the Conventional PCI mode, when the data word which was target aborted on the internal bus is actually requested and delivered on the PCI Bus, and the ATU ECC Target Abort Enable bit is set in the ATUIMR, a target abort is returned to the PCI initiator on that data word. When the ATU ECC Target Abort Enable bit is cleared in the ATUIMR, a disconnect with data is returned to the PCI initiator during the data word that was target aborted on the internal bus. In both cases, the IRQ is flushed after the completion cycle is performed on the PCI bus

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI target interface during an inbound read transaction:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
- When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.

2.7.9.2.2 PCI-X Mode

When operating in the PCI-X mode, a Target-Abort of an inbound read transaction (split read request) on the Internal Bus results in the following actions.

- The ATU initiates a Split Completion Error Message (with message class=2h - completer error and message index=81h - 81341 and 81342 internal bus target abort) on the PCI bus.
- When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.

Note: This split completion error message includes a device specific message index. The error handler would need to have knowledge of the device specific error messages of the 81341 and 81342 in order to fully diagnose the problem.



2.7.9.3 Parity Error on the Internal Bus

The 81341 and 81342 provides support for byte-wise parity protection on the internal bus. The internal bus consists of a 36 bit address bus and 128 bit data bus; both are protected by byte-wise parity. The internal bus parity protection is provided independent of the operating mode of the ATUs PCI interface.

When initiating transactions on the internal bus, the ATUs internal bus interface generates byte-wise parity. As a target the ATU checks byte-wise parity.

As an initiator, for outbound write transactions the ATU will forward bad data parity on the PCI bus if the ATU detected a parity error on the internal bus interface. And for outbound read transactions the ATU will forward bad data parity on the internal bus if the ATU detected a parity error on the PCI interface. Outbound read parity error will be detected and logged by the internal bus initiator. For outbound read data that has to flow through the internal bus bridge, the bridge will log the error. Refer to the internal bus bridge chapter for more details on how the parity error is handled.

As a target, for inbound read transactions the ATU will forward bad data parity on the PCI bus if the ATU detected a parity error on the internal bus interface. And for inbound write transactions the ATU will forward bad data parity on the internal bus if the ATU detected a parity error on the PCI interface. Inbound write parity error will be detected and logged by the internal bus target. For write data that has to flow through the internal bus bridge, the bridge will log the error. Refer to the internal bus bridge chapter for more details on how the parity error is handled.

2.7.9.3.1 Conventional Mode

On an inbound read transaction, when the data word where the internal bus parity error is detected is actually requested and returned to the PCI bus, a target abort is returned to the PCI initiator on that data word. The IRQ is flushed after the completion cycle is performed on the PCI bus

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI target interface during an inbound read transaction:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
- When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.

2.7.9.3.2 PCI-X Mode

An internal bus parity error of an inbound read transaction (split read request) on the Internal Bus results in the following actions.

- The ATU initiates a Split Completion Error Message (with message class=2h - completer error and message index=81h - 81341 and 81342 internal bus target abort) on the PCI bus.
- When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.

Note: This split completion error message includes a device specific message index. The error handler would need to have knowledge of the device specific error messages of the 81341 and 81342 in order to fully diagnose the problem.



2.7.10 ATU Error Summary

Table 16 summarizes the ATU error reporting for PCI bus errors and Table 17 summarizes the ATU error reporting for internal bus errors. The tables assume that all error reporting is enabled through the appropriate command registers (unless otherwise noted). The ATU Status Register records PCI bus errors. Note that the **SERR#** Asserted bit in the Status Register is set only when the **SERR#** Enable bit in the Command Register is set. The ATU Interrupt Status Registers record Intel XScale® processor interrupt status information.

Table 16. ATU Error Reporting Summary - PCI Interface (Sheet 1 of 5)

Error Condition (Bus Mode ^a)	Bits Set in ATU Status Register (ATUSR ^b) or PCI-X Status Register (PCIXSR ^c) and/or ECC Logging Registers ^d (ECCLOG)		Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Uncorrectable Address or Attribute Error (All)	In Conventional Mode, ignore (Master-Abort) the transaction, and then signal SERR# . In PCI-X Mode, claim the transaction and complete as when no error had occurred; and, signal SERR# upon uncorrectable error detection.		
(All)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(All)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(All)	Detected Uncorrectable Address or Attribute Error - bit 20 of the PCI Configuration and Status Register	N/A	N/A
(All)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
(PCI-X2)	ECCLOG Updated	N/A	N/A
Correctable Address or Attribute Error (PCI-X2)	The transaction is completed as when no error had occurred. Then the transaction is forwarded to the internal bus normally.		
(PCI-X2)	ECCLOG Updated	Detected Correctable Error - bit 14	ATUIMR bit 11
Outbound Read Request Uncorrectable Data Error (All)	Signal PERR# and SERR# (PCI-X Mode Only).		
(All)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(All)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Outbound Write Request Uncorrectable Data Error (All)	Signal SERR# (only for PCI-X or MSI Writes).		
(All)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X or MSI)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(PCI-X or MSI)	N/A	SERR# Detected - bit 4	ATUCR bit 9
Inbound Read Completion Uncorrectable Data Error (PCI-X)	None.		

**Table 16. ATU Error Reporting Summary - PCI Interface (Sheet 2 of 5)**

Error Condition (Bus Mode ^a)	Bits Set in ATU Status Register (ATUSR ^b) or PCI-X Status Register (PCIXSR ^c) and/or ECC Logging Registers ^d (ECCLOG)	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Inbound Configuration Write Completion Message Uncorrectable Data Error (PCI-X)	None.		
Inbound Read Request Uncorrectable Data Error (All)	None.		
Inbound Write Request Uncorrectable Data Error (All)	Signal PERR# .		
(All)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
(PCI-X2)	ECCLOG Updated	N/A	N/A
Outbound Read Completion Uncorrectable Data Error (All)	Signal PERR# and SERR# .		
(PCI-X)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(PCI-X)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
(PCI-X2)	ECCLOG Updated	N/A	N/A
Outbound Split Write Uncorrectable Data Error Message (PCI-X)	Signal SERR# .		
(PCI-X)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(PCI-X)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Inbound Configuration Write Request Uncorrectable Data Error (All)	Signal PERR# . Initiate an Uncorrectable Split Write Data Error Message addressed to the Requester (PCI-X Mode Only).		
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10
(All)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
(PCI-X2)	ECCLOG Updated	N/A	N/A
Split Completion Message Uncorrectable Data Error (PCI-X)	Signal PERR# and SERR# .		



Table 16. ATU Error Reporting Summary - PCI Interface (Sheet 3 of 5)

Error Condition (Bus Mode ^a)	Bits Set in ATU Status Register (ATUSR ^b) or PCI-X Status Register (PCIXSR ^c) and/or ECC Logging Registers ^d (ECCLOG)	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
(PCI-X)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(PCI-X and SCE ^e)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
(PCI-X)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
(PCI-X2)	ECCLOG Updated	N/A	N/A
Outbound Read Request Correctable Data Error (PCI-X2)	None.		
Outbound Write Request Correctable Data Error (PCI-X2)	None.		
Inbound Read Completion Correctable Data Error (PCI-X)	None.		
Inbound Configuration Write Completion Message Correctable Data Error (PCI-X)	None.		
Inbound Read Request Correctable Data Error (PCI-X2)	None.		
Inbound Write Request Correctable Data Error (PCI-X2)	None.		
(PCI-X2)	ECCLOG Updated	Detected Correctable Error - bit 14	ATUIMR bit 11
Outbound Read Completion Correctable Data Error (PCI-X2)	None.		
(PCI-X2)	ECCLOG Updated	Detected Correctable Error - bit 14	ATUIMR bit 11
Inbound Configuration Write Request Correctable Data Error (PCI-X2)	None.		
(PCI-X2)	ECCLOG Updated	Detected Correctable Error - bit 14	ATUIMR bit 11

**Table 16. ATU Error Reporting Summary - PCI Interface (Sheet 4 of 5)**

Error Condition (Bus Mode ^a)	Bits Set in ATU Status Register (ATUSR ^b) or PCI-X Status Register (PCIXSR ^c) and/or ECC Logging Registers ^d (ECCLOG)	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Split Completion Message Correctable Data Error (PCI-X)	None.		
(PCI-X2)	ECCLOG Updated	Detected Correctable Error - bit 14	ATUIMR bit 11
Outbound Read Request Master- Abort (All)	None.		
(All)	Master Abort - bit 13	PCI Master Abort - bit 3	ATUIMR bit 5
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Outbound Write Request Master- Abort (All)	None.		
(All)	Master Abort - bit 13	PCI Master Abort - bit 3	ATUIMR bit 5
(MSI)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(MSI)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Inbound Read Completions Master-Abort (PCI-X)	None.		
Inbound Configuration Write Completion Message Master- Abort (PCI-X)	None.		
Outbound Read Request Target- Abort (All)	None.		
(All)	Target Abort (master) - bit 12	PCI Target Abort (master) - bit 2	ATUIMR bit 4
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Outbound Write Request Target- Abort (All)	None.		
(All)	Target Abort (master) - bit 12	PCI Target Abort (master) - bit 2	ATUIMR bit 4
(MSI)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(MSI)	N/A	SERR# Detected - bit 4	ATUCR bit 9
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Inbound EROM Write Request Target-Abort (All)	Signal Target-Abort.		



Table 16. ATU Error Reporting Summary - PCI Interface (Sheet 5 of 5)

Error Condition (Bus Mode ^a)	Bits Set in ATU Status Register (ATUSR ^b) or PCI-X Status Register (PCIXSR ^c) and/or ECC Logging Registers ^d (ECCLG)	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
(All)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
Unexpected Split Completion (PCI-X)	In the PCI-X mode, the transaction completes normally according to the invalid lower address field or invalid byte count.		
(PCI-X)	Unexpected Split Completion - bit 19	N/A	N/A

- Codes for bus mode in which this error response applies: PCI-X means PCI-X Mode 1 or PCI-X Mode 2, PCI-X2 means PCI-X Mode 2 only, Conventional means Conventional PCI Mode Only, and All means that the error response applies in the Conventional, PCI-X Mode 1 and PCI-X Mode 2 modes of operation. MSI stands for Message-Signaled Interrupts and refers to an Outbound Write transaction that is actually an MSI write transaction.
- Table assumes that Parity Error Response - bit 6 of the ATUCMD register is set.
- Table assumes that Data Parity Recovery Enable - bit 0 of the PCIXCMD is clear.
- When a correctable or uncorrectable data error occurs in PCI-X Mode 2, the ECC Logging registers consisting of the [ECC Control and Status Register - ECCCSR](#), the [ECC First Address Register - ECCFAR](#), the [ECC Second Address Register - ECCSAR](#), and the [ECC Attribute Register - ECCAR](#) are updated with additional information about the ECC error.
- When the SCE bit (bit 30 of the Completer Attributes) and the SCM bit (bit 29 of the Completer Attributes) are set during the Attribute phase of a Split Completion Transaction, the transaction is a Split Completion Message that is an Error Message. In this case, the Received Split Completion Error Message - bit 29 of the PCIXSR is set.

**Table 17. ATU Error Reporting Summary - Internal Bus Interface**

Error Condition ^a (Bus Mode ^b)	Bits Set in ATU Status Register (ATUSR ^c)	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)			
Inbound Write Request Master-Abort (All)	Assert SERR# .		
(All)	N/A	Internal Bus Master Abort - bit 7	N/A
(All)	SERR# Asserted - bit 14	SERR# Asserted - bit 10	ATUIMR bit 6
(All)	N/A	SERR# Detected - bit 4	ATUCR bit 9
Inbound Read Request Master-Abort (All)	In the Conventional Mode signal Target-Abort. In the PCI-X Mode send a device specific Split Completion Error Message to the Requester.		
(All)	N/A	Internal Bus Master Abort - bit 7	N/A
(Conventional)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10
Inbound Read Request Target-Abort (All)	In the Conventional Mode signal Target-Abort. In the PCI-X Mode send a device specific Split Completion Error Message to the Requester.		
(Conventional)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10

- a. There are no Inbound Write Request Target-Abort Error Conditions.
- b. Codes for bus mode in which this error response applies: PCI-X means PCI-X Mode 1 or PCI-X Mode 2, PCI-X2 means PCI-X Mode 2 only, Conventional means Conventional PCI Mode Only, and All means that the error response applies in the Conventional, PCI-X Mode 1 and PCI-X Mode 2 modes of operation. MSI stands for Message-Signaled Interrupts and refers to an Outbound Write transaction that is actually an MSI write transaction.
- c. Table assumes that the ATU Inbound **SERR#** Enable bit (bit 1 of the ATUIMR), the ATU ECC Target Abort Enable (bit 0 of the ATUIMR), and the **SERR#** Enable bit (bit 8 of the ATUCMD) are set.



2.8 Message-Signaled Interrupts

The Messaging Unit is responsible for the generation of all of the Outbound Interrupts from the 81341 and 81342. These interrupts can be delivered to the Host Processor via the **P_INTA#** output pin or the Message Signaled Interrupt (MSI) mechanism.

When a host processor enables Message-Signaled Interrupts (MSI) on the 81341 and 81342, an outbound interrupt is signaled to the host via a PCI write instead of the assertion of the **P_INTA#** output pin.

In support of MSI, the 81341 and 81342 implements the MSI capability structure. The capability structure includes the [Section 4.9.30, "MSI Capability Identifier Register - Cap_ID" on page 450](#), the [Section 4.9.31, "MSI Next Item Pointer Register - MSI_Next_Ptr" on page 451](#), the [Section 4.9.33, "Message Address Register - Message_Address" on page 453](#), the [Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454](#) and the [Section 4.9.35, "Message Data Register- Message_Data" on page 455](#).

The Message Unit generates MSIs by writing to the MSI port via the internal bus. The ATU generates a write transaction whenever the Message Unit writes to the MSI port, using the address specified in the [Section 4.9.33, "Message Address Register - Message_Address" on page 453](#), the [Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454](#) and the [Section 4.9.35, "Message Data Register- Message_Data" on page 455](#).



2.9 Internal Interrupts

The ATU has 3 internal interrupts that connect to the internal Interrupt Controller Unit.

- ATU Interrupt Status Register Interrupt
- ATU Configuration Write Interrupt
- ATU BIST Interrupt



2.10 Vital Product Data

Vital Product Data (VPD) provides detailed information to the system regarding the hardware, software and microcode elements of a device. This information may include Part Number, Serial Number or other detailed information. This information resides on a non-volatile storage device (i.e., Flash Memory) attached to the 81341 and 81342. In addition VPD also provides a mechanism for storing information such as performance or failure data on the device being monitored.

Support of VPD involves the implementation of the VPD Extended Capabilities List Item in the Primary ATU. The VPD Extended capabilities header consists of five registers, the “VPD Capability Identifier Register - VPD_Cap_ID” on page 190, the “VPD Next Item Pointer Register - VPD_Next_Item_Ptr” on page 190, the “VPD Address Register - VPDAR” on page 191, and the “VPD Data Register - VPDDR” on page 191.

Scheduled by Intel XScale® processor interrupts, the 81341 and 81342 may be used to retrieve or store VPD information through the VPD extended capabilities list item.

Please consult Appendix I of the *PCI Local Bus Specification*, Revision 2.3 for the definitions of compliant VPD format.

2.10.1 Configuring Vital Product Data Operation

By default, the 81341 and 81342 VPD functionality is not configured for operation. Specifically, the VPD Extended Capabilities List Item is not discovered during a PCI bus scan and the ATUs VPD interrupt status bit in the “ATU Interrupt Status Register - ATUISR” on page 186 is masked by the “ATU Interrupt Mask Register - ATUIMR” on page 188. The following steps should be followed to properly configure the 81341 and 81342 support for VPD:

1. The 81341 and 81342 must be strapped to Retry Type 0 Configuration cycles following the deassertion of **P_RST#**. Enabling this configuration cycle retry mechanism insures that the Intel XScale® processor can make the VPD Extended Capabilities List Item visible before the system configures the 81341 and 81342. The configuration retry mechanism is controlled through bit 2 of the “PCI Configuration and Status Register - PCSR” on page 183.
2. When the configuration retry mechanism is strapped enabled as described in step 1, typically, the 81341 and 81342 would also be strapped such that the Intel XScale® processor would immediately boot following the deassertion of **P_RST#** (bit 1 of the PCSR), though this is not required.
3. The Intel XScale® processor writes E8H to the “PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr” on page 196. This links the PCI-X Capabilities List Item to the VPD Capabilities List Item.
4. The Intel XScale® processor clears bit 12 of the ATUIMR to enable the ATUs VPD interrupt status bit.



2.10.2 Accessing Vital Product Data

The VPD Capabilities List Item provides three fields which the system uses to access the Vital Product Data:

VPD Address	DWORD Aligned Byte address of the VPD to be accessed which is represented by VPDAR[14:0]. Note that this means that the maximum size of the VPD is 128 Kbytes. The user may pick any 128 Kbyte block of memory in the storage component for the VPD.
Flag	The flag register is used to indicate when the transfer between the VPD Data Register and the storage component is completed. The flag is in VPDAR[15] which means that the Flag is written at the same time that VPD address is written.
VPD Data	Four bytes of VPD Data can be read or written through this field which is represented by VPDDR[31:0]. The least significant byte of this register represents the byte at the VPD Address (VPDAR[14:0]). Four bytes are always transferred between this register and the VPD storage component.

2.10.2.1 Reading Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 81341 and 81342 reads Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag cleared.
2. An interrupt to the Intel XScale® processor is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be set.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being set, the results of the original read operation are unpredictable.

3. Using the VPD Address, the Intel XScale® processor retrieves the Vital Product Data from the VPD storage component (i.e., Flash Memory).
4. The Intel XScale® processor then writes this data to VPD Data Register (VPDDR).
5. The Intel XScale® processor clears the VPD interrupt status bit in the ATUISR.
6. The Intel XScale® processor then sets the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been set, the host processor then reads the retrieved VPD from the VPDDR.



2.10.2.2 Writing Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 81341 and 81342 writes Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD data to be written to the VPDDR.
2. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag set.
3. An interrupt to the Intel XScale® processor is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be cleared.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being cleared, the results of the original write operation are unpredictable.

4. Using the VPD Address, the Intel XScale® processor writes the Vital Product Data from the VPDDR to the VPD storage component (i.e., Flash Memory).
5. The Intel XScale® processor clears the VPD interrupt status bit in the ATUISR.
6. The Intel XScale® processor then clears the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been cleared, the host processor has been informed that the VPD write operation is complete.



2.11 Central Resource Functionality

The ATUs PCI interface provides the ability to initialize and control a PCI bus segment. The ATU includes the Central Resource functionality necessary for the ATU to serve this role in the most efficient and cost-effective manner for up to four PCI devices besides the 81341 and 81342. The Central Resource features include the delivery of PCI Reset, Secondary Clock Outputs, Bus Mode and Frequency initialization, and a PCI Bus Arbiter.

2.11.1 Multi-Function Support

When operating as a central resource, the ATU behaves as a single function device and claims memory transactions only for the ATU function. This means that the BARs and control signals from other internal functions are ignored, and any PCI status updates are limited to the ATU.

2.11.2 Inbound Transactions

When operating as a central resource, the ATU does not accept I/O transactions. The ATU accepts inbound configuration transactions. However, as the Central Resource, the ATU should not issue configuration transactions to itself. Another option is to permanently pull the IDSEL pin low on the board to prevent the ATU from claiming configuration transactions. Transactions are claimed only when they map into one of the inbound memory BARs.

2.11.3 Outbound Transactions

ATU outbound transaction support is detailed in [Section 2.2.2, "Outbound Transactions-Single Address Cycle \(SAC\) Internal Bus Transactions"](#). The behavior is the same in both end point and central resource modes of operation.

2.11.4 PCI Reset (P_RSTOUT#)

When the Central Resource is enabled (**PCIX_EP#** = 1), the ATU controls generation of **P_RSTOUT#** for the PCI Bus segment attached to the ATU.

The ATU controls the PCI Reset signal (**P_RSTOUT#**) for the external agents supported by the 81341 and 81342 Central Resource. When the Central Resource is enabled, **P_RSTOUT#** remains asserted following the deassertion of the fundamental reset. **P_RSTOUT#** can be deasserted in the ["PCI Configuration and Status Register - PCSR"](#) on [page 183](#).

2.11.5 PCI Clock Outputs (P_CLKOUT, P_CLKO[3:0])

The **P_CLKOUT/P_CLKO[3:0]** PCI clock outputs are available when operating in central resource mode and the primary clock source is the PCI Express* reference clock (**CLK_SRC_PCIE#** = 0). In this operating mode, the **P_CLKOUT** signal is connected to the **P_CLKIN** pin and trace match to the **P_CLKO[3:0]** signals.

Warning: When the **P_CLKIN** is the primary clock source (**CLK_SRC_PCIE#** = 1), the PCI Clock outputs are disabled and cannot be used as a clock source for any device.



2.11.6 External Clock Driver (CR_FREQ[1:0])

When the internal PCI Clock outputs are not sufficient, an external clock driver can be used to supply additional PCI Clocks. To facilitate the use of an external driver, the **CR_FREQ[1:0]** pins are driven based on the settings in the PCI-X capability field (bits 19:16) in the “[PCI Configuration and Status Register - PCSR](#)”. These output pins can be connected to a PCI-X clock driver to select the desired bus frequency.

Table 18. CR_FREQ[1:0] Encoding

PCSR[19:16] (PCIX Init Pattern)	PCSR[10] (P_M66EN)	CR_FREQ[1:0]	Bus Frequency	Bus Mode
1111	0	11	33 MHz	Conventional PCI
1111	1	10	66 MHz	Conventional PCI
1110	-	10	66 MHz	PCI-X
1101	-	01	100 MHz	PCI-X
1100	-	00	133 MHz	PCI-X



2.11.7 Bus Mode and Frequency Initialization

The ATUs PCI Bus interface is capable of operating at a variety of frequencies, and in either Conventional PCI mode, or in PCI-X mode. The bus mode is established when coming out of the bus segment reset sequences. When the ATUs central resource is enabled, the resultant mode and frequency is dependent upon the device capabilities reported as well as any system specific loading information.

The ATU, as the central resource is the originating device for the PCI bus and as such, sets the bus mode and frequency when exiting out of the bus reset sequence. The two key components that factor into the resultant secondary bus mode and frequency are the PCI-X standard sampling of downstream device capabilities, and the system specific physical bus loading characteristics for which the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 does not provide any standard means of reporting.

Downstream device capabilities are indicated by the values of **P_M66EN**, and **P_PCIXCAP** during **P_RST#** assertion.

Table 19. Device Mode/Frequency Capability Reporting

M66EN	PCIXCAP ^a	Conventional PCI Device Frequency Capability	PCI-X Device Frequency Capability
Ground	Ground	33 MHz	Not capable
8.2K pull-up ^b	Ground	66 MHz	Not capable
Ground	10K Pull-down	33 MHz	PCI-X 66 MHz
8.2K pull-up ^b	10K Pull-down	66 MHz	PCI-X 66 MHz
Ground	NC	33 MHz	PCI-X 133 MHz
8.2K pull-up ^b	NC	66 MHz	PCI-X 133 MHz
Ground	3.16K 1% pull-down	33 MHz	PCI-X 266 MHz
8.2K pull-up ^b	3.16K 1% pull-down	66 MHz	PCI-X 266 MHz

a. Resistor values are specified in Section 2.3.4, "PCIXCAP and MODE2 Connection," in *PCI-X Electrical and Mechanical Addendum to the PCI Local Bus Specification*, Revision 2.0.

b. M66EN may be pulled high on the motherboard.

Note: Knowledge of the device capabilities alone is insufficient information to robustly select the bus frequency. In order to be sure of what the bus operating frequency should be set to, knowledge of the bus layout (e.g., number of slots), is necessary.

When, for example, a 133 MHz PCI-X capable adapter was the sole occupant of a two slot segment, then it would be necessary to slow the bus to 100 MHz, even though the card reported it could operate at 133 MHz due to the additional electrical loading imposed by the two slot board and connector layout.

The ATU provides a strapping approach for reporting system specific bus loading information that is used in determining the maximum operating frequency of the secondary bus. The ATU considers this strap along with the device capabilities reported during **P_RST#** to determine the PCI bus's mode and frequency when emerging from **P_RST#**.

This strap, entitled PCI-X Bus 100 MHz Enable, is sampled on **PCIXM1_100#**, indicating to the ATU what to limit the bus frequency to a maximum of 100MHz. The value of this field is determined by the system designer, after having assessed the characteristics of the PCI bus system/adaptor implementation.



Table 20 details the PCI bus frequency initialization as a function of the PCI Bus **PCIXM1_100#** and **PCIXM2_100#** reset strap, and the sampled secondary device capabilities when operating in PCI-X mode.

Table 20. PCI Bus Frequency Initialization^a

P_PCIXCAP	P_MODE 2	P_M66EN	PCIXM1_100#	PCIXM2_100#	PCI Bus Mode	PCI Bus Frequency	PCSR[19:16]
< 0.11VCC	- ^b	Ground	-	-	PCI	33 MHz	1111
< 0.11VCC	-	Not connected	-	-	PCI	66 MHz	1111
<0.6VCC & >0.11 VCC	GND	-	-	-	PCI-X Mode 1	66 MHz	1110
<0.6VCC & >0.11VCC	VCC	-	-	GND	PCI-X Mode 2	100 MHz (PCI-X 200 MHz)	0101
<0.6VCC & >0.11VCC	VCC	-	-	VCC	PCI-X Mode 2	133 MHz (PCI-X 266 MHz)	0100
<0.89VCC & >0.6VCC	-	-	-	-	PCI-X Mode 1	66 MHz	1110
>0.89VCC	-	-	GND	-	PCI-X Mode 1	100 MHz	1101
>0.89VCC	-	-	VCC	-	PCI-X Mode 1	133 MHz	1100

a. 81341 and 81342 does not support PCI-X 533 Mhz.

b. A '-' in the table indicates the value is a don't care for computing the bus mode/frequency. All signals must still be pulled to a valid logic level.

Note:

The PCI Bus PCI-X 100 MHz Enable strapping feature enables implementations to force the PCI bus of 81341 and 81342 to operate at 100 MHz even with no standard provisions in the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 for reporting device capability of 100 MHz operation.

When a card is plugged into a four slot PCI bus, a **P_PCIXCAP** pull-down (R1) strapping ensures that the bus runs at no greater than 66 MHz in PCI-X mode, and grounding **P_M66EN** ensures that the bus runs at no greater than 33 MHz in PCI, regardless of the reported downstream device capabilities.

When a card is plugged into a two slot secondary bus, the **PCIXM1_100#** pull-down strapping ensures that the bus runs at no greater than 100 MHz in PCI-X mode regardless of the reported downstream device capabilities.

When a card is plugged into a single slot secondary (i.e., a segment that should be able to run at 133 MHz), by strapping the **PCIXM1_100#** to 0b (as though it were a two slot configuration), the bus operates at 100 MHz maximum⁶.

6. Adapters that report 133MHz PCI-X device capability with this **PCIXM1_100#** setting is limited to 100MHz operation.



Table 21 describes the bus mode and frequency initialization pattern that the ATU signals on its secondary bus when coming out of **P_RST#**, after having evaluated the above information.

Table 21. PCI-X Initialization Pattern

PERR#	DEVSEL#	STOP#	TRDY#	Mode ^a	Clock Period (Ns)		Clock Frequency (MHz)	
					Maximum	Minimum	Minimum	Maximum
Deasserted	Deasserted	Deasserted	Deasserted	PCI 33	60	30	16	33
				PCI 66	30	15	33	66
Deasserted	Deasserted	Deasserted	Asserted	PCI-X Mode 1	20	15	50	66
Deasserted	Deasserted	Asserted	Deasserted	PCI-X Mode 1	15	10	66	100
Deasserted	Deasserted	Asserted	Asserted	PCI-X Mode 1	10	7.5	100	133
Deasserted	Asserted	Deasserted	Deasserted	PCI-X Mode 1	Reserved			
Deasserted	Asserted	Deasserted	Asserted	PCI-X Mode 1				
Deasserted	Asserted	Asserted	Deasserted	PCI-X Mode 1				
Deasserted	Asserted	Asserted	Asserted	PCI-X Mode 1				
Asserted	Deasserted	Deasserted	Deasserted	PCI-X 266 (Mode 2)				
Asserted	Deasserted	Deasserted	Asserted	PCI-X 266 ^b (Mode 2)	20	15	50	66
Asserted	Deasserted	Asserted	Deasserted	PCI-X 266 (Mode 2)	15	10	66	100
Asserted	Deasserted	Asserted	Asserted	PCI-X 266 (Mode 2)	10	7.5	100	133
Asserted	Asserted	Deasserted	Deasserted	PCI-X	Reserved			
Asserted	Asserted	Deasserted	Asserted	PCI-X				
Asserted	Asserted	Asserted	Deasserted	PCI-X				
Asserted	Asserted	Asserted	Asserted	PCI-X				

a. 81341 and 81342 supports neither PCI-X 533 Mode nor ECC in Mode 1.

b. The **P_CLK[3:0]** frequency and associated initialization pattern in PCI-X 266 mode for is selected in the “PCI Configuration and Status Register - PCSR” on page 183.

When operating in Central Resource Mode (**PCIX_EP#** = 1), the PCI-X initialization pattern is driven directly from the PCI-X capability field (bits 19:16) in the “PCI Configuration and Status Register - PCSR”. The default value of this field is determined by the **PCIX_EP#**, **PCIXM1_100#**, **PCIXM2_100#** straps, as well as the **P_MODE2**, **P_PCIXCAP**, and **P_M66EN**, signals as described in Table 20, “PCI Bus Frequency Initialization” on page 138.

While **P_RSTOUT#** is asserted the initialization pattern is driven on the PCI bus. Software can override the default pattern by writing a new value to PCSR[19:16], before clearing the Central Resource PCI Bus Reset field (bit 21) in the “PCI Configuration and Status Register - PCSR”.

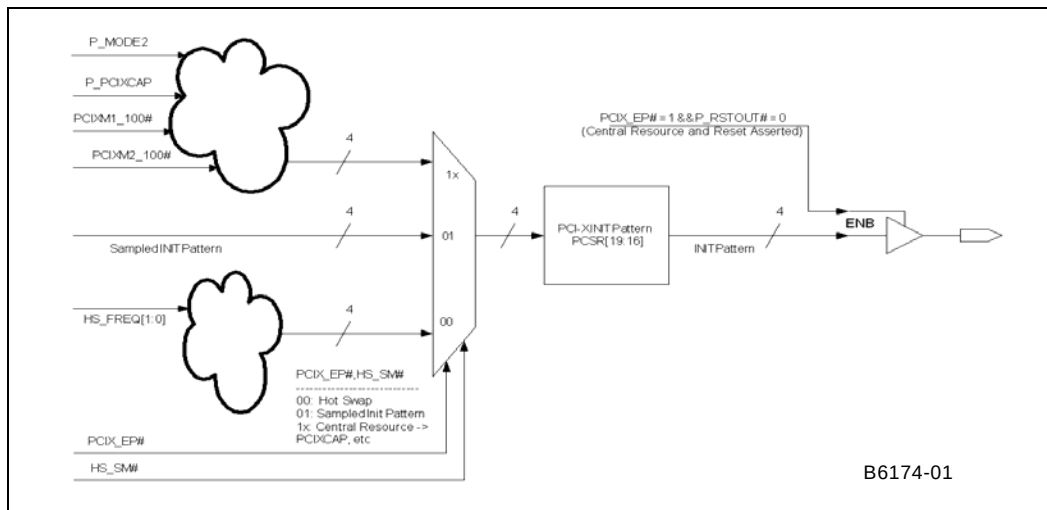
When **P_RSTOUT#** is asserted any time after initial power on, software must enforce the 1ms reset assertion time (T_{rst}) required in the PCI specifications.

When operating as an endpoint (**PCIX_EP#** = 0), PCSR[19:16] contains the initialization pattern captured off the bus during **P_RST#**.



When operating as an endpoint in Hot-Swap mode (**HS_SM#** = 0), **PCSR[19:16]** is set based on the **HS_FREQ[1:0]** pins. For more details see [Table 6, “HS_FREQ Encoding”](#) on page 86.

Figure 14. PCI-X Initialization Pattern Setting / Drive



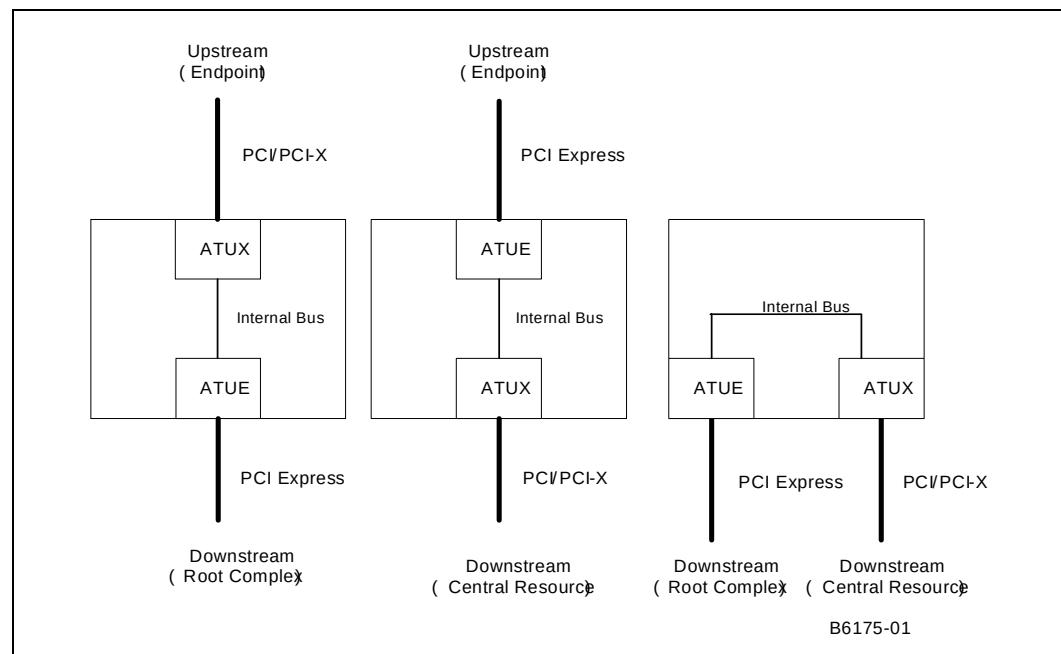


2.12 Embedded Bridge Functionality

The dual interface 81341 and 81342 provides two Address Translation Units: the ATUE provides a PCI Express interface, and the ATUX provides a PCI/PCIX interface. The two ATUs are internally connected via the south internal bus.

The two ATUs can be setup and used as an embedded bridge (non-transparent bridge). For example, the dual interface 81341 and 81342 can be used to bridge a PCI Express device and a PCI-X device in a system. Both ATUs support endpoint mode and root complex or central resource mode capabilities. Therefore, either ATU can be connected upstream or downstream in a system. [Figure 15](#) shows the possible configurations of the dual interface 81341 and 81342.

Figure 15. Intel® 81341 and 81342 I/O Processors PCI Interface Configuration Examples



To allow PCI transactions to flow between the two PCI interfaces of the dual interface 81341 and 81342 in a bidirectional manner, each of the ATUs must have at least one of its Inbound Address Translation Windows mapped to one of the other ATU's Outbound Address Translation Windows. Refer to [Section 2.2.2, "Outbound Transactions- Single Address Cycle \(SAC\) Internal Bus Transactions"](#) on page 72 that explain how the Inbound and Outbound address translation windows operate. Below are few examples that demonstrate how the ATUs may be programmed to operate as an embedded bridge.

**Example 2. ATUE as an Endpoint and ATUX as a Central Resource**

The user wants to setup the ATUE as an endpoint and ATUX as a central resource by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 9 and the ATUX outbound window is placed in section 8 of the 64-GByte internal address space.

- External Straps: `INTERFACE_SEL_PCIX# = '1'`, `PCIE_RC# = '1'`, and `PCIX_EP# = '1'`. Note that setting up `INTERFACE_SEL_PCIX#` to '1' causes the ATUE to be function number 0H and the ATUX to be function number 5H. The function number must be consistent with the function number field programmed in the `OUMBAR2[30:28]` since this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: `IALR2 = 00000000H`, `IATVR2 = 00000000H`, `IAUTVR2 = 00000008H`.
- Outbound Address Translation Window for ATUX: `OUMBAR2 = D0000008H`, `OUMWTVR2 = 00000000H`
- Inbound Address Translation Window for ATUX: `IALR2 = 00000000H`, `IATVR2 = 00000000H`, `IAUTVR2 = 00000009H`.
- Outbound Address Translation Window for ATUE: `OUMBAR2 = 80000009H`, `OUMWTVR2 = 00000000H`.

Example 3. ATUE as a Root Complex and ATUX as an Endpoint

The user wants to setup the ATUE as root complex and ATUX as an endpoint by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 7 and the ATUX outbound window is placed in section 6 of the 64-GByte internal address space.

- External Straps: `INTERFACE_SEL_PCIX# = '0'`, `PCIE_RC# = '0'`, and `PCIX_EP# = '0'`. Note that setting up `INTERFACE_SEL_PCIX#` to '0' causes the ATUX to be function number 0H and the ATUE to be function number 5H. The function number must be consistent with the function number field programmed in the `OUMBAR2[30:28]` since this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: `IALR2 = 00000000H`, `IATVR2 = 00000000H`, `IAUTVR2 = 00000006H`.
- Outbound Address Translation Window for ATUX: `OUMBAR2 = 80000006H`, `OUMWTVR2 = 00000000H`
- Inbound Address Translation Window for ATUX: `IALR2 = 00000000H`, `IATVR2 = 00000000H`, `IAUTVR2 = 00000007H`.
- Outbound Address Translation Window for ATUE: `OUMBAR2 = D0000007H`, `OUMWTVR2 = 00000000H`.



Example 4. ATUE as a Root Complex and ATUX as a Central Resource

The user wants to setup the ATUE as root complex and ATUX as a central resource by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 6 and the ATUX outbound window is placed in section 9 of the 64-GByte internal address space.

- External Straps: `INTERFACE_SEL_PCIX#` = '0', `PCIE_RC#` = '0', and `PCIX_EP#` = '1'. Note that setting up `INTERFACE_SEL_PCIX#` to '0' causes the ATUX to be function number 0H and the ATUE to be function number 5H. In this scenario since non of the ATUs is an endpoint, `INTERFACE_SEL_PCIX#` can be setup to either '0' or '1'. However, the function number must be consistent with the function number field programmed in the `OUMBAR2[30:28]` since this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: `IALR2` = 00000000H, `IATVR2` = 00000000H, `IAUTVR2` = 00000009H.
- Outbound Address Translation Window for ATUX: `OUMBAR2` = 80000009H, `UMWTVR2` = 00000000H
- Inbound Address Translation Window for ATUX: `IALR2` = 00000000H, `IATVR2` = 00000000H, `IAUTVR2` = 00000006H.
- Outbound Address Translation Window for ATUE: `OUMBAR2` = D0000006H, `UMWTVR2` = 00000000H.

Warning: The following restrictions on the embedded bridge transactions from the ATUX to ATUE need to be noted.

- When the ATUX is operating in PCI-X mode, Inbound Write transactions with non-contiguous byte enables are not supported. Note that byte enables may be disabled in the first and/or last DWORD of an Inbound Write transaction.
- When the ATUX is operating in conventional PCI mode, all of the byte enables must be asserted on Inbound Write transactions.



2.13 Register Definitions

Every PCI device implements its own separate configuration address space and configuration registers. The *PCI Local Bus Specification*, Revision 2.3 requires that configuration space be 256 bytes, and the first 64 bytes must adhere to a predefined header format.

Figure 16 defines the header format. Table 22 shows the PCI configuration registers, listed by internal bus address offset. Table 22 shows the entire ATU configuration space (including header and extended registers) and the corresponding section that describes each register. Note that all configuration read and write transactions is accepted on the internal bus as 32-bit transactions. Refer to Chapter 21.0, “Peripheral Registers”.

2.13.1 PCI Configuration Registers

Figure 16. ATU Interface Configuration Header Format

ATU Device ID		Vendor ID		00H
Status		Command		04H
ATU Class Code			Revision ID	08H
ATUBISTR	Header Type	Latency Timer	Cacheline Size	0CH
Inbound ATU Base Address 0				10H
Inbound ATU Upper Base Address 0				14H
Inbound ATU Base Address 1				18H
Inbound ATU Upper Base Address 1				1CH
Inbound ATU Base Address 2				20H
Inbound ATU Upper Base Address 2				24H
Reserved				28H
ATU Subsystem ID		ATU Subsystem Vendor ID		2CH
Expansion ROM Base Address				30H
Reserved			Capabilities Pointer	34H
Reserved				38H
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	3CH

The ATU is programmed via a Type 0 configuration command on the PCI interface. See [Section 2.2.1.4, “Inbound Configuration Cycle Translation” on page 69](#). ATU configuration space is function number zero of the 81341 and 81342 single-function PCI device.

Beyond the required 64 byte header format, ATU configuration space implements extended register space in support of the units functionality. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details on accessing and programming configuration register space.

The ATU unit includes six extended capability configuration spaces beginning at configuration offsets 90H, 98H, A0H, B0H, D0H, and E8H. The extended configuration spaces can be accessed by a device on the PCI interface through a mechanism defined in the *PCI Local Bus Specification*, Revision 2.3.



In the ATU Status Register (Section 2.13.6) the appropriate bit is set indicating that the Extended Capability Configuration space is supported. When this bit is read, the device can then read the Capabilities Pointer register (Section 2.13.22) to determine the configuration offset of the Extended Capabilities Configuration Header. The format of these headers are depicted in Figure 17, Figure 19, Figure 20, Figure 20 and Figure 22.

Figure 17. ATU Interface Extended Configuration Header Format (Power Management)

Power Management Capabilities	Next Item Pointer	Capability Identifier	98H
Reserved	Power Management Control/Status		9CH

B6326-01

The first byte at the Extended Configuration Offset 98H is the ATU Capability Identifier Register (Section). This identifies this Extended Configuration Header space as the type defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 2.13.49) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to B0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

To enable the *PCI Bus Power Management Interface Specification*, Revision 1.1 compliance support, the Power State Transition interrupt mask in bit 8 of the ATUIMR needs to be cleared. It is the configuration software's responsibility to properly enable and initialize the ATUs Power Management Interface before the Configuration Cycle Retry Bit in the Section 2.13.41, "PCI Configuration and Status Register - PCSR" on page 183 is cleared in order for the ATU to be *Advanced Configuration and Power Interface Specification*, Revision 2.0 compliant.

Figure 18. ATU Interface Extended Configuration Header Format (MSI-X Capability)

MSI-X Message Control	MSI-X Next Item Pointer	MSI-X Capability ID	B0H
MSI-X Table Offset		Table BIR	B4H
MSI-X PBA Offset		PBA BIR	B8H
B6327-01			

Note: MSI-X Capability Registers are defined in Chapter 4.0, "Messaging Unit."

The first byte at the Extended Configuration Offset B0H is the MSI-X Capability Identifier Register (Section 4.9.36, "MSI-X Capability Identifier Register - MSI-X_Cap_ID"). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.



Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 4.9.37, “MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr”) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to A0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 19. ATU Interface Extended Configuration Header Format (MSI Capability)

MSI Message Control	MSI Next Item Pointer	MSI Capability ID	A0H
MSI Message Address			A4H
MSI Message Upper Address			A8H
Reserved	MSI Message Data		ACH

B6328-01

Note: MSI-X Capability Registers are defined in Chapter 4.0, “Messaging Unit.”

The first byte at the Extended Configuration Offset A0H is in Section 4.9.30, “MSI Capability Identifier Register - Cap_ID”. This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Section 4.9.31, “MSI Next Item Pointer Register - MSI_Next_Ptr”, which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to D0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 20. ATU Interface Extended Configuration Header Format (PCI-X Capability Type 1)

PCI-X Command	Next Item Pointer	PCI-X Capability ID	D0H
PCI-X Status			D4H
ECC Control and Status (Mode 2)			D8H
ECC First Address (Mode 2)			DCH
ECC Second Address (Mode 2)			E0H
ECC Attribute (Mode 2)			E4H

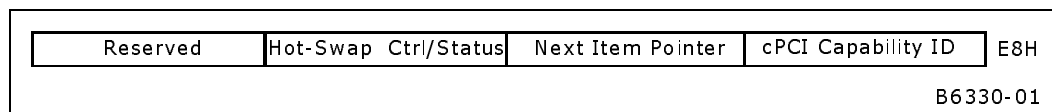
B6329-01

The first byte at the Extended Configuration Offset D0H is the PCI-X Capability Identifier Register (Section 2.13.53). This identifies this Extended Configuration Header space as the type defined by the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0. Indicated by bits 13:12 of the PCI-X Command Register, the 81341 and 81342 includes version 1 of the PCI-X Capabilities List Item indicating the 24 byte length of the item and the fact that ECC protection is provided in Mode 2 only.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 2.13.54) which indicates configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to E8H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.



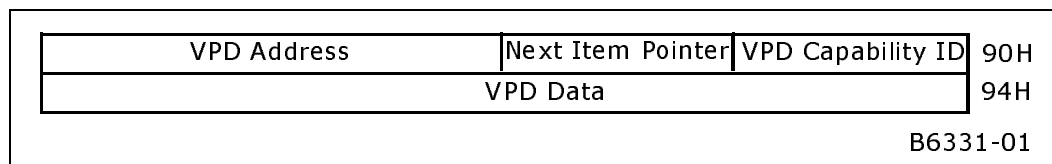
Figure 21. ATU Extended Configuration Header Format (Compact PCI Hot-Swap Capability)



The first byte at the Extended Configuration Offset E8H is the Compact PCI Hot-Swap Capability Identifier Register (Section 2.13.61). This identifies this Extended Configuration Header space as the type defined by the *Compact PCI Hot-Swap Specification*, Revision 2.1.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 2.13.62) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H by default to indicate there are no additional Extended Capabilities Headers in the ATU configuration space. Software can set this pointer to 90H indicating there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 22. ATU Interface Extended Configuration Header Format (VPD Capability)



The first byte at the Extended Configuration Offset 90H is the VPD Capability Identifier Register (Section 2.13.44). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 2.13.45) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H indicating that there are no additional Extended Capabilities Headers supported in the ATUs configuration space.

The following sections describe the ATU and Expansion ROM configuration registers. Configuration space consists of 8, 16, 24, and 32-bit registers arranged in a predefined format. Each register is described in functionality, access type (read/write, read/clear, read only) and reset default condition.

All registers adhere to the definitions found in the *PCI Local Bus Specification*, Revision 2.3 unless otherwise noted.

The Register Offset for PCI configuration registers is given in Table 22. As stated, a Type 0 configuration command on the bus, with an active **IDSEL** or a memory-mapped internal bus access required to read or write these registers. The ATU is always located at Function 0 in configuration space.

PCI Configuration command access to registers with offsets higher than 0FFH is not supported.

Note: In PCI-X Mode 2, the ATU handles all configuration command accesses to registers with offsets higher than 0FFH as Reserved.

Note: Each configuration register access type is individually defined for PCI configuration accesses. Some PCI read-only configuration registers have read/write capability from the 81341 and 81342 core CPU. See also Chapter 21.0, "Peripheral Registers".



2.13.2 Internal Bus Registers

A subset of the ATU registers are accessible through both inbound PCI configuration cycles and the 81341 and 81342 core CPU (Register offsets 000H through 0FFH). The balance of the registers are accessible only via the internal bus. [Table 22, “Address Translation Unit Registers” on page 148](#) represents all of the ATU registers.

The location of these registers are specified as a relative offset to a 512KB aligned global PMMR offset. The default for the 512KB aligned offset is 0 FFD8 0000H defined by the PMMRBAR register. See also [Chapter 21.0, “Peripheral Registers”](#).

The Internal Bus Address Offset to PMMRBAR of any ATU Register can be derived by adding the 4 KB address aligned Internal Bus Memory Mapped Register Range Offset ([Table 23, “ATU Internal Bus Memory Mapped Register Range Offsets” on page 151](#)) to the Register Offset ([Table 22, “Address Translation Unit Registers” on page 148](#))

For example, when **INTERFACE_SEL_PCIX#** is asserted, the offset to PMMRBAR of the “ATU Command Register - ATUCMD” would be (4 C000H+004H) or 4 C004H.

Note: The 4 KB Address Aligned Range Offset can be different depending on two configuration straps as described in [Table 23](#).

Table 22. Address Translation Unit Registers (Sheet 1 of 3)

Register Offset	ATU Register Section, Name, Page
000H	Section 2.13.3, “ATU Vendor ID Register - ATUVID” on page 152
002H	Section 2.13.4, “ATU Device ID Register - ATUDID” on page 152
004H	Section 2.13.5, “ATU Command Register - ATUCMD” on page 153
006H	Section 2.13.6, “ATU Status Register - ATUSR” on page 154
008H	Section 2.13.7, “ATU Revision ID Register - ATURID” on page 156
009H	Section 2.13.8, “ATU Class Code Register - ATUCCR” on page 156
00CH	Section 2.13.9, “ATU Cacheline Size Register - ATUCLSR” on page 157
00DH	Section 2.13.10, “ATU Latency Timer Register - ATULT” on page 157
00EH	Section 2.13.11, “ATU Header Type Register - ATUHTR” on page 158
00FH	Section 2.13.12, “ATU BIST Register - ATUBISTR” on page 159
010H	Section 2.13.13, “Inbound ATU Base Address Register 0 - IABAR0” on page 160
014H	Section 2.13.14, “Inbound ATU Upper Base Address Register 0 - IAUBAR0” on page 161
018H	Section 2.13.15, “Inbound ATU Base Address Register 1 - IABAR1” on page 162
01CH	Section 2.13.16, “Inbound ATU Upper Base Address Register 1 - IAUBAR1” on page 163
020H	Section 2.13.17, “Inbound ATU Base Address Register 2 - IABAR2” on page 164
024H	Section 2.13.18, “Inbound ATU Upper Base Address Register 2 - IAUBAR2” on page 165
02CH	Section 2.13.19, “ATU Subsystem Vendor ID Register - ASVIR” on page 166
02EH	Section 2.13.20, “ATU Subsystem ID Register - ASIR” on page 166
030H	Section 2.13.21, “Expansion ROM Base Address Register - ERBAR” on page 167
034H	Section 2.13.22, “ATU Capabilities Pointer Register - ATU_Cap_Ptr” on page 168
03CH	Section 2.13.24, “ATU Interrupt Line Register - ATUILR” on page 171
03DH	Section 2.13.25, “ATU Interrupt Pin Register - ATUIPR” on page 172
03EH	Section 2.13.26, “ATU Minimum Grant Register - ATUMGNT” on page 172
03FH	Section 2.13.27, “ATU Maximum Latency Register - ATUMLAT” on page 173
040H	Section 2.13.28, “Inbound ATU Limit Register 0 - IALR0” on page 174
044H	Section 2.13.29, “Inbound ATU Translate Value Register 0 - IATVR0” on page 175
048H	Section 2.13.30, “Inbound ATU Upper Translate Value Register 0 - IAUTVR0” on page 175

**Table 22. Address Translation Unit Registers (Sheet 2 of 3)**

Register Offset	ATU Register Section, Name, Page
04CH	Section 2.13.31, "Inbound ATU Limit Register 1 - IALR1" on page 176
050H	Section 2.13.32, "Inbound ATU Translate Value Register 1 - IATVR1" on page 177
054H	Section 2.13.33, "Inbound ATU Upper Translate Value Register 1 - IAUTVR1" on page 177
058H	Section 2.13.34, "Inbound ATU Limit Register 2 - IALR2" on page 178
05CH	Section 2.13.35, "Inbound ATU Translate Value Register 2 - IATVR2" on page 179
060H	Section 2.13.36, "Inbound ATU Upper Translate Value Register 2 - IAUTVR2" on page 179
064H	Section 2.13.37, "Expansion ROM Limit Register - ERLR" on page 180
068H	Section 2.13.38, "Expansion ROM Translate Value Register - ERTVR" on page 181
06CH	Section 2.13.39, "Expansion ROM Upper Translate Value Register - ERUTVR" on page 181
070H	Section 2.13.40, "ATU Configuration Register - ATUCR" on page 182
074H	Section 2.13.41, "PCI Configuration and Status Register - PCSR" on page 183
078H	Section 2.13.42, "ATU Interrupt Status Register - ATUISR" on page 186
07CH	Section 2.13.43, "ATU Interrupt Mask Register - ATUIMR" on page 188
080H — 08FH	Reserved
090H	Section 2.13.44, "VPD Capability Identifier Register - VPD_Cap_ID" on page 190
091H	Section 2.13.45, "VPD Next Item Pointer Register - VPD_Next_Item_Ptr" on page 190
092H	Section 2.13.46, "VPD Address Register - VPDAR" on page 191
094H	Section 2.13.47, "VPD Data Register - VPDDR" on page 191
098H	Section 2.13.48, "PM Capability Identifier Register - PM_Cap_ID" on page 192
099H	Section 2.13.49, "PM Next Item Pointer Register - PM_Next_Item_Ptr" on page 192
09AH	Section 2.13.50, "ATU Power Management Capabilities Register - APMCR" on page 193
09CH	Section 2.13.51, "ATU Power Management Control/Status Register - APMCSR" on page 194
0A0H	Section 4.9.30, "MSI Capability Identifier Register - Cap_ID" on page 450 ^a
0A1H	Section 4.9.31, "MSI Next Item Pointer Register - MSI_Next_Ptr" on page 451
0A2H	Section 4.9.32, "Message Control Register - Message_Control" on page 452
0A4H	Section 4.9.33, "Message Address Register - Message_Address" on page 453
0A8H	Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454
0ACH	Section 4.9.35, "Message Data Register - Message_Data" on page 455
0AEH	Reserved
0B0H	Section 4.9.36, "MSI-X Capability Identifier Register - MSI-X_Cap_ID" on page 456
0B1H	Section 4.9.37, "MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr" on page 457
0B2H	Section 4.9.38, "MSI-X Message Control Register - MSI-X_MCR" on page 458
0B4H	Section 4.9.39, "MSI-X Table Offset Register — MSI-X_Table_Offset" on page 459
0B8H	Section 4.9.40, "MSI-X Pending Bit Array Offset Register - MSI-X_PBA_Offset" on page 460
0BCH — 0C8H	Reserved
0CCH	Section 2.13.52, "ATU Scratch Pad Register - ATUSPR" on page 195
0D0H	Section 2.13.53, "PCI-X Capability Identifier Register - PCI-X_Cap_ID" on page 195
0D1H	Section 2.13.54, "PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr" on page 196
0D2H	Section 2.13.55, "PCI-X Command Register - PCIXCMD" on page 196
0D4H	Section 2.13.56, "PCI-X Status Register - PCIXSR" on page 198
0D8H	Section 2.13.57, "ECC Control and Status Register - ECCCSR" on page 200
0DCH	Section 2.13.58, "ECC First Address Register - ECCFAR" on page 203
0E0H	Section 2.13.59, "ECC Second Address Register - ECCSAR" on page 204
0E4H	Section 2.13.60, "ECC Attribute Register - ECCAR" on page 205
0E8H	Section 2.13.61, "CompactPCI Hot-Swap Capability ID Register" on page 205

**Table 22. Address Translation Unit Registers (Sheet 3 of 3)**

Register Offset	ATU Register Section, Name, Page
0E9H	Section 2.13.62, "Offset EDh: HS_NXTP - Next Item Pointer" on page 206
0EAH	Section 2.13.63, "HS_CNTRL - Hot-Swap Control/Status Register" on page 207
0EBH	Reserved
0ECH — 0FFH	Reserved
100H — 1FFH	Reserved
200H	Section 2.13.64, "Inbound ATU Base Address Register 3 - IABAR3" on page 209
204H	Section 2.13.65, "Inbound ATU Upper Base Address Register 3 - IAUBAR3" on page 210
208H	Section 2.13.66, "Inbound ATU Limit Register 3 - IALR3" on page 211
20CH	Section 2.13.67, "Inbound ATU Translate Value Register 3 - IATVR3" on page 212
210H	Section 2.13.68, "Inbound ATU Upper Translate Value Register 3 - IAUTVR3" on page 212
214H — 2FCH	Reserved
300H	Section 2.13.69, "Outbound I/O Base Address Register - OIOBAR" on page 213
304H	Section 2.13.70, "Outbound I/O Window Translate Value Register - OIOWTVR" on page 214
308H	Section 2.13.71, "Outbound Upper Memory Window Base Address Register 0 - OUMBAR0" on page 215
30CH	Section 2.13.72, "Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0" on page 216
310H	Section 2.13.73, "Outbound Upper Memory Window Base Address Register 1 - OUMBAR1" on page 217
314H	Section 2.13.74, "Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1" on page 218
318H	Section 2.13.75, "Outbound Upper Memory Window Base Address Register 2 - OUMBAR2" on page 219
31CH	Section 2.13.76, "Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2" on page 220
320H	Section 2.13.77, "Outbound Upper Memory Window Base Address Register 3 - OUMBAR3" on page 221
324H	Section 2.13.78, "Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3" on page 222
328H	Reserved
32CH	Reserved
330H	Section 2.13.79, "Outbound Configuration Cycle Address Register - OCCAR" on page 223
334H	Section 2.13.80, "Outbound Configuration Cycle Data Register - OCCDR" on page 224
338H	Section 2.13.81, "Outbound Configuration Cycle Function Number - OCCFN" on page 224
33CH — 37CH	Reserved
380H	Section 2.13.82, "PCI Interface Error Control and Status Register - PIECSR" on page 225
384H	Section 2.13.83, "PCI Interface Error Address Register - PCIEAR" on page 226
388H	Section 2.13.84, "PCI Interface Error Upper Address Register - PCIEUAR" on page 227
38CH	Section 2.13.85, "PCI Interface Error Context Address Register — PCIECAR" on page 228
394H	Section 2.13.86, "Internal Arbiter Control Register - IACR" on page 229
398H	Section 2.13.87, "Multi-Transaction Timer - MTT" on page 230
39CH	Reserved.

- a. All MSI and MSI-X capability register descriptions are in Section 4.9, "Register Definitions" on page 425 of Chapter 4.0, "Messaging Unit."

**Table 23. ATU Internal Bus Memory Mapped Register Range Offsets**

INTERFACE_SEL_PCIX#	Internal Bus MMR Address Range Offset (Relative to PMMRBAR)
Asserted (0)	+4 8000H
Deasserted (1)	+4 C000H

Table 24. PCI-X Pad Registers

Register Offset	Section, Register Name - Acronym (Page)
2100H	Section 2.13.88, "PCIX RCOMP Control Register — PRCR" on page 231
2104H	Section 2.13.89, "PCIX Pad ODT Drive Strength Manual Override Values Registers — PPODSMOVR" on page 232
2108H	Section 2.13.90, "PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V/1.5 V Switch Supply Voltage) — PPDSMOVR3.3_1.5" on page 233
210CH	Section 2.13.91, "PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V Dedicated Supply Voltage) — PPDSMOVR3.3" on page 234



2.13.3 ATU Vendor ID Register - ATUVID

ATU Vendor ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 25. ATU Vendor ID Register - ATUVID

Register Offset +000H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:00	8086H	ATU Vendor ID - This is a 16-bit value assigned to Intel. This register, combined with the DID, uniquely identify the PCI device. Access type is Read/Write to allow the 81341 and 81342 to configure the register as a different vendor ID to simulate the interface of a standard mechanism currently used by existing application software.

2.13.4 ATU Device ID Register - ATUDID

ATU Device ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 26. ATU Device ID Register - ATUDID

Register Offset +002H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:00	Product Type Dependent ^a	ATU Device ID - This is a 16-bit value assigned to the ATU. This ID, combined with the VID, uniquely identify any PCI device.

a. See Intel® 81341 and 81342 I/O Processors *Specification Update*.



2.13.5 ATU Command Register - ATUCMD

ATU Command Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3 and in most cases, affect the behavior of the PCI ATU and devices on the PCI bus.

Table 27. ATU Command Register - ATUCMD

Register Offset +004H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:11	00000 ₂	Reserved
10	0 ₂	Interrupt Disable - Setting this bit disables the ATU function of 81341 and 81342 from asserting the INT[A:D]# signals. Clearing this bit enables the assertion of the INT[A:D]# signals by the ATU function of 81341 and 81342.
09	0 ₂	Fast Back to Back Enable - When cleared, the ATU interface is not allowed to generate fast back-to-back cycles on its bus. Ignored when operating in the PCI-X mode.
08	0 ₂	SERR# Enable - When cleared, the ATU interface is not allowed to assert SERR# on the PCI interface.
07	0 ₂	Hard-wired 0
06	0 ₂	Parity Error Response - When set, the ATU takes normal action when a parity (or uncorrectable ECC) error is detected. When cleared, parity (or uncorrectable ECC) checking is disabled. Note: When the bit is cleared in PCI-X Mode 2, the 81341 and 81342 records all ECC errors in the ECC Control and Status, ECC First Address, ECC Second Address, and ECC Attribute registers but in all other respects treats the transaction as though it had no error. Correctable ECC errors are corrected independent of the state of this bit.
05	0 ₂	VGA Palette Snoop Enable - The ATU interface does not support I/O writes and therefore, does not perform VGA palette snooping.
04	0 ₂	Memory Write and Invalidate Enable - When set, ATU may generate MWI commands. When clear, ATU use Memory Write commands instead of MWI. Ignored when operating in the PCI-X mode.
03	0 ₂	Special Cycle Enable - The ATU interface does not respond to special cycle commands in any way. Not implemented and a reserved bit field.
02	0 ₂	Bus Master Enable - The ATU interface can act as a master on the PCI bus. When cleared, disables the device from generating PCI accesses. When set, allows the device to behave as a PCI bus master. When operating in the PCI-X mode, ATU initiates a split completion transaction regardless of the state of this bit.
01	0 ₂	Memory Enable - Controls the ATU interface's response to PCI memory addresses. When cleared, the ATU interface does not respond to any memory access on the PCI bus.
00	0 ₂	I/O Space Enable - Controls the ATU interface response to I/O transactions. The ATU does not support I/O transactions. This bit is Read Only.



2.13.6 ATU Status Register - ATUSR

The ATU Status Register bits adhere to the *PCI Local Bus Specification*, Revision 2.3 definitions. The *read/clear* bits can only be set by internal hardware and cleared by either a reset condition or by writing a 1₂ to the register.

Table 28. ATU Status Register - ATUSR (Sheet 1 of 2)

15 12 8 4 0 IOP Attributes rc rc rc rc rc ro ro rc ro ro ro ro ro rv rv rv PCI Attributes rc rc rc rc rc ro ro rc ro ro ro ro ro rv rv rv Register Offset +006H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear RO = Read Only PR = Preserved RS = Read/Set NA = Not Accessible		
Bit	Default	Description
15	0 ₂	Detected Parity Error - set when a parity error is detected in data received by the ATU on the PCI bus even when the ATUCMD register's Parity Error Response bit is cleared. Set under the following conditions: - Uncorrectable Write Data Error when the ATU is a target (inbound write). - Uncorrectable Read Data Error when the ATU is a requester (outbound read). - Any Uncorrectable Address or Attribute (PCI-X Only) Error on the Bus (including one generated by the ATU).
14	0 ₂	SERR# Asserted - set when SERR# is asserted on the PCI bus by the ATU.
13	0 ₂	Master Abort - set when a transaction initiated by the ATU PCI master interface, ends in a Master-Abort or when the ATU receives a Master Abort Split Completion Error Message in PCI-X mode.
12	0 ₂	Target Abort (master) - set when a transaction initiated by the ATU PCI master interface, ends in a target abort or when the ATU receives a Target Abort Split Completion Error Message in PCI-X mode.
11	0 ₂	Target Abort (target) - set when the ATU interface, acting as a target, terminates the transaction on the PCI bus with a target abort.
10:09	01 ₂	DEVSEL# Timing - These bits are read-only and define the slowest DEVSEL# timing for a target device in Conventional PCI Mode regardless of the operating mode (except configuration accesses). 00₂ = Fast 01₂ = Medium 10₂ = Slow 11₂ = Reserved The ATU interface uses Medium timing.
08	0 ₂	Master Parity Error - The ATU interface sets this bit under the following conditions: - The ATU asserted PERR# itself or the ATU observed PERR# asserted. - And the ATU acted as the requester for the operation in which the error occurred. - And the ATUCMD register's Parity Error Response bit is set - Or (PCI-X Mode Only) the ATU received an Uncorrectable Write Data Error Message - And the ATUCMD register's Parity Error Response bit is set
07	1 ₂ (Conventional mode) 0 ₂ (PCI-X mode)	Fast Back-to-Back - The ATU interface is capable of accepting fast back-to-back transactions in Conventional PCI mode when the transactions are not to the same target. Since fast back-to-back transactions do not exist in PCI-X mode, this bit is forced to 0 in the PCI-X mode.
06	0 ₂	UDF Supported - User Definable Features are not supported
05	1 ₂	66 MHz. Capable - 66 MHz operation is supported.
04	1 ₂	Capabilities - When set, this function implements extended capabilities.

**Table 28. ATU Status Register - ATUSR (Sheet 2 of 2)**

Register Offset +006H IOP Attributes PCI Attributes Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
03	0 ₂	Interrupt Status - This bit reflects the state of the interrupt in the 81341 and 81342 ATU function. Only when the Interrupt Disable bit in the ATUCMD is a 0 and this Interrupt Status bit is a 1, are any of the 81341 and 81342's INT[A:D]# signals asserted by the ATU function. Note: Setting the Interrupt Disable bit to a 1 in (bit 10 of ATUCMD) has no effect on the state of this bit.
02:00	000 ₂	Reserved



2.13.7 ATU Revision ID Register - ATURID

Revision ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 29. ATU Revision ID Register - ATURID

Register Offset +008H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	xxH ^a	ATU Revision - identifies the revision number.

a. See Intel® 81341 and 81342 I/O Processors Specification Update.

2.13.8 ATU Class Code Register - ATUCCR

Class Code Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. Auto configuration software reads this register to determine the PCI device function.

Table 30. ATU Class Code Register - ATUCCR

Register Offset +009H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
23:16	05H	Base Class - Memory Controller
15:08	80H	Sub Class - Other Memory Controller
07:00	00H	Programming Interface - None defined



2.13.9 ATU Cacheline Size Register - ATUCLSR

Cacheline Size Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register is programmed with the system cacheline size in DWORDs (32-bit words). Cacheline Size is restricted to either 0, 8 or 16 DWORDs; the ATU interprets any other value as "0".

Table 31. ATU Cacheline Size Register - ATUCLSR

Register Offset +00CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	ATU Cacheline Size - specifies the system cacheline size in DWORDs. Cacheline size is restricted to either 0, 8 or 16 DWORDs.

2.13.10 ATU Latency Timer Register - ATULT

ATU Latency Timer Register bit definitions apply to the PCI interface.

Table 32. ATU Latency Timer Register - ATULT

Register Offset +00DH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:03	00000 ₂ (for Conventional mode) 01000 ₂ (for PCI-X mode)	Programmable Latency Timer - This field varies the latency timer for the interface from 0 to 248 clocks. The default value is 0 clocks for Conventional PCI mode, and 64 clocks for PCI-X mode.
02:00	000 ₂	Latency Timer Granularity - These Bits are read only giving a programmable granularity of 8 clocks for the latency timer.



2.13.11 ATU Header Type Register - ATUHTR

Header Type Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register indicates the layout of ATU configuration space bytes 10H to 3FH. The MSB indicates whether or not the device is multi-function.

Table 33. ATU Header Type Register - ATUHTR

Register Offset +00EH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07	End Point: DF_SEL[2:0] = "000" Central Resource: 0	Single Function/Multi-Function Device - Identifies the 81341 and 81342 as a single-function or multi-function PCI device depending on the setting of the DF_SEL[2:0] strap during P_RST# assertion. Note: The 81341 and 81342 can be configured as a single-function device (ATU only) or a multi-function device (ATU and storage controller) for split driver support. As a Central Resource, this bit is always '0'.
06:00	000000 ₂	PCI Header Type - This bit field indicates the type of PCI header implemented. The ATU interface header conforms to <i>PCI Local Bus Specification</i> , Revision 2.3.



2.13.12 ATU BIST Register - ATUBISTR

The ATU BIST Register controls the functions the Intel XScale® processor performs when BIST is initiated. This register is the interface between the host processor requesting BIST functions and the 81341 and 81342 replying with the results from the software implementation of the BIST functionality.

Table 34. ATU BIST Register - ATUBISTR

Register Offset +00FH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07	0 ₂	BIST Capable - This bit value is always equal to the ATUCR ATU BIST Interrupt Enable bit. See Section 2.13.40, "ATU Configuration Register - ATUCR" on page 182 .
06	0 ₂	Start BIST - When the ATUCR BIST Interrupt Enable bit is set: Setting this bit generates an interrupt to the Intel XScale® processor to perform a software BIST function. The Intel XScale® processor clears this bit when the BIST software has completed with the BIST results found in ATUBISTR register bits [3:0]. When the ATUCR BIST Interrupt Enable bit is clear: Setting this bit does not generate an interrupt to the Intel XScale® processor and no BIST functions is performed. The Intel XScale® processor does not clear this bit.
05:04	00 ₂	Reserved
03:00	0000 ₂	BIST Completion Code - when the ATUCR BIST Interrupt Enable bit is set and the ATUBISTR Start BIST bit is set (bit 6): The Intel XScale® processor places the results of the software BIST in these bits. A nonzero value indicates a device-specific error.



The Inbound ATU Base Address Register 0 (IABAR0) together with the Inbound ATU Upper Base Address Register 0 (IAUBAR0) defines the block of memory addresses where the inbound translation window 0 begins. The inbound ATU decodes and forwards the bus request to the 81341 and 81342 internal bus with a translated address to map into 81341 and 81342 local memory. The IABAR0 and IAUBAR0 define the base address and describes the required memory block size; see [Section 2.13.23, “Determining Block Sizes for Base Address Registers” on page 169](#). Bits 31 through 12 of the IABAR0 is either read/write bits or read only with a value of 0 depending on the value located within the IALR0. This configuration allows the IABAR0 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

- a. Since non-prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is cleared prior to host configuration, the user should also set the Type Indicator for 32-bit addressability.
- b. For compliance to the *PCI-X Protocol Addendum to the PCI Local Bus Specification, Revision 2.0*, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64-bit addressability. This is the default for IABAR0.

Register Offset +010H

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



2.13.14 Inbound ATU Upper Base Address Register 0 - IAUBAR0

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type Indicator of IABAR0 is set to indicate 32 bit addressability, the IAUBAR0 register attributes are read-only. Prior to changing the Type Indicator in the IABAR0 to support 32-bit addressability, the IAUBAR0 must be written with zero unless it already contains zero. Zero is the default value for the IAUBAR0.

Table 36. Inbound ATU Upper Base Address Register 0 - IAUBAR0

		IOP Attributes PCI Attributes 312824201612840 </																															



2.13.16 Inbound ATU Upper Base Address Register 1 - IAUBAR1

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR1 is set to indicate 32 bit addressability, the IAUBAR1 register attributes are read-only. By default the IAUBAR1 register has read-only attributes. Prior to changing the Type Indicator in the IABAR1 to support 32-bit addressability, the IAUBAR1 must be written with zero unless it already contains zero. Zero is the default value for IAUBAR1.

Table 38. Inbound ATU Upper Base Address Register 1 - IAUBAR1

Register Offset +01CH		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													



The Inbound ATU Base Address Register 2 (IABAR2) together with the Inbound ATU Upper Base Address Register 2 (IAUBAR2) defines the block of memory space addresses where the inbound translation window 2 begins. The inbound ATU decodes and forwards the bus request to the 81341 and 81342 internal bus with a translated address to map into 81341 and 81342 local memory. The IABAR2 and IAUBAR2 (Memory Space only) define the base address and describes the required address block size; see [Section 2.13.23, “Determining Block Sizes for Base Address Registers” on page 169](#). Bits 31 through 8 of the IABAR2 is either read/write bits or read only with a value of 0 depending on the value located within the IALR2. This configuration allows the IABAR2 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

- e. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is cleared prior to host configuration, the user should also set the Type Indicator for 32 bit addressability.
- f. For compliance to the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability. This is the default for IABAR2.

Register Offset +020H

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- RC = Read Clear
- RO = Read Only
- RS = Read/Set
- NA = Not Accessible



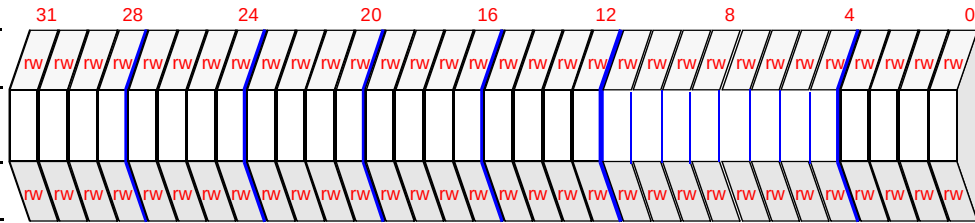
2.13.18 Inbound ATU Upper Base Address Register 2 - IAUBAR2

This register contains the upper base address when decoding PCI addresses for memory space (Memory Space Indicator in IABAR2 is clear) beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR2 is set to indicate 32 bit addressability, the IAUBAR2 register attributes are read-only. By default the IAUBAR2 register has read-only attributes. Prior to changing the Type Indicator in the IABAR2 to support 32-bit addressability, the IAUBAR2 must be written with zero unless it already contains zero. Zero is the default value for IAUBAR2.

Table 40. Inbound ATU Upper Base Address Register 2 - IAUBAR2

																														
Register Offset +024H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																												
Bit	Default	Description																												
31:0	00000H	Translation Upper Base Address 2 - Together with the Translation Base Address 2 these bits define the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes.																												



2.13.19 ATU Subsystem Vendor ID Register - ASVIR

ATU Subsystem Vendor ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 41. ATU Subsystem Vendor ID Register - ASVIR

1512840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Register Offset +02CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:0	0000H	Subsystem Vendor ID - This register uniquely identifies the add-in board or subsystem vendor.

2.13.20 ATU Subsystem ID Register - ASIR

ATU Subsystem ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 42. ATU Subsystem ID Register - ASIR

1512840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Register Offset +02EH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:0	0000H	Subsystem ID - uniquely identifies the add-in board or subsystem.

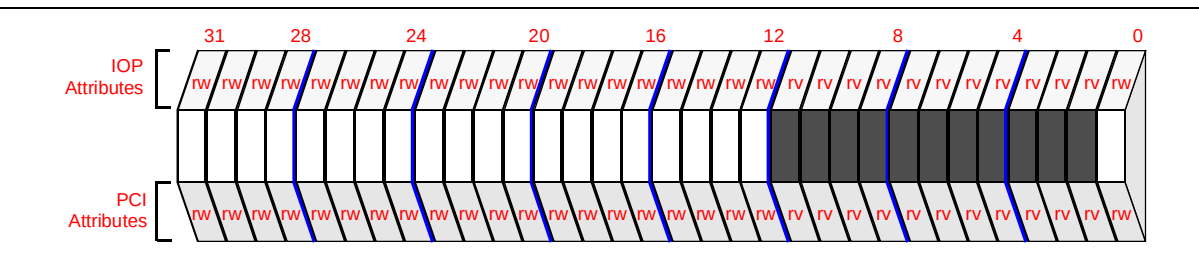


2.13.21 Expansion ROM Base Address Register - ERBAR

The Expansion ROM Base Address Register defines the block of memory addresses used for containing the Expansion ROM. It permits the inclusion of multiple code images, allowing the device to be initialized. The code image supplied consists of either executable code or an interpreted code. Each code image must start on a 512 byte boundary and each must contain the PCI Expansion ROM header. Image placement in ROM space depends on the length of code images which precede it within ROM. ERBAR defines the base address and describes the required memory block size; see [Section 2.13.23](#). Expansion ROM address space (limit size) can be a maximum of 16 MBytes. Bits 31 through 12 of the ERBAR is either read/write bits or read only with a value of 0 depending on the value located within the ERLR. This configuration allows the ERBAR to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The Expansion ROM Base Address Register's programmed value must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming Expansion ROM base address registers.

Table 43. Expansion ROM Base Address Register -ERBAR

		
Register Offset +030H	PCI Configuration Address Offset 30H - 33H	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:12	00000H	Expansion ROM Base Address - These bits define the actual location where the Expansion ROM address window resides when addressed from the PCI bus on any 4 Kbyte boundary.
11:01	000H	Reserved
00	0 ₂	Address Decode Enable - This bit field shows the ROM address decoder is enabled or disabled. When cleared, indicates the address decoder is disabled.



2.13.22 ATU Capabilities Pointer Register - ATU_Cap_Ptr

The Capabilities Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register provides an offset in this function's PCI Configuration Space for the location of the first item in the first Capability list. In the case of the 81341 and 81342, this is the PCI Bus Power Management extended capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 44. ATU Capabilities Pointer Register - ATU_Cap_Ptr

Register Offset +034H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
36 Bit	Default	Description
07:00	98H	Capability List Pointer - This provides an offset in this function's configuration space that points to the 81341 and 81342 PCI Bus Power Management extended capability.



2.13.23 Determining Block Sizes for Base Address Registers

The required address size and type can be determined by writing ones to a base address register and reading from the registers. By scanning the returned value from the least-significant bit of the base address registers upwards, the programmer can determine the required address space size. The binary-weighted value of the first non-zero bit found indicates the required amount of space. Table 45 describes the relationship between the values read back and the byte sizes the base address register requires.

Note: The user must exercise caution when re-programming the BAR, Limit, and Translate Value Registers. Since these 3 registers can not be programmed simultaneously, it is recommended that the BAR be disabled during reprogramming.

Table 45. Memory Block Size Read Response

Response After Writing all 1s to the Base Address Register	Size (in Bytes)	Response After Writing all 1s to the Base Address Register	Size (in Bytes)
FFFFFFF0H	16	FFF00000H	1 M
FFFFFFE0H	32	FFE00000H	2 M
FFFFFFC0H	64	FFC00000H	4 M
FFFFFF80H	128	FF800000H	8 M
FFFFFF00H	256	FF000000H	16 M
FFFFFE00H	512	FE000000H	32 M
FFFFFC00H	1K	FC000000H	64 M
FFFFF800H	2K	F8000000H	128 M
FFFFF000H	4K	F0000000H	256 M
FFFFE000H	8K	E0000000H	512 M
FFFFC000H	16K	C0000000H	1 G
FFFF8000H	32K	80000000H	2 G
FFFF0000H	64K	00000000H	Register not implemented, no address space required.
FFFE0000H	128K		
FFFC0000H	256K		
FFF80000H	512K		

As an example, assume that FFFF.FFFFH is written to the ATU Inbound Base Address Register 0 (IABAR0) and the value read back is FFF0.0008H. Bit zero is a zero, so the device requires memory address space. Bit three is one, so the memory does support prefetching. Scanning upwards starting at bit four, bit twenty is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicated that the device requires 1 Mbyte of memory space.

The ATU Base Address Registers and the Expansion ROM Base Address Register use their associated limit registers to enable which bits within the base address register are read/write and which bits are read only (0). This allows the programming of these registers in a manner similar to other PCI devices even though the limit is variable.

**Table 46. ATU Base Registers and Associated Limit Registers**

Base Address Register	Limit Register ^a	Description
Inbound ATU Base Address Register 0	Inbound ATU Limit Register 0	Defines the inbound translation window 0 from the PCI bus.
Inbound ATU Upper Base Address Register 0	N/A	Together with ATU Base Address Register 0 defines the inbound translation window 0 from the PCI bus for DACs.
Inbound ATU Base Address Register 1	Inbound ATU Limit Register 1	Defines the inbound translation window 1 from the PCI bus.
Inbound ATU Upper Base Address Register 1	N/A	Together with ATU Base Address Register 1 defines the inbound translation window 1 from the PCI bus for DACs.
Inbound ATU Base Address Register 2	Inbound ATU Limit Register 2	Defines the inbound translation window 2 from the PCI bus.
Inbound ATU Upper Base Address Register 2	N/A	Together with ATU Base Address Register 2 defines the inbound translation window 2 from the PCI bus for DACs.
Inbound ATU Base Address Register 3	Inbound ATU Limit Register 3	Defines the inbound translation window 3 from the PCI bus.
Inbound ATU Upper Base Address Register 3	N/A	Together with ATU Base Address Register 3 defines the inbound translation window 3 from the PCI bus for DACs. Note: This is a private BAR that resides outside of the standard PCI configuration header space (offsets 00H-3FH).
Expansion ROM Base Address Register	Expansion ROM Limit Register	Defines the window of addresses used by a bus master for reading from an Expansion ROM.

- a. For Inbound Memory Windows 0-2, bit 0 of the limit register is a “claiming disable” bit. This feature allows the Memory Window to be used to a Memory Range for use in communication with Private PCI devices.



2.13.24 ATU Interrupt Line Register - ATUILR

ATU Interrupt Line Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the system interrupt controller's interrupt request lines which connect to the device's PCI interrupt request lines (as specified in the interrupt pin register).

In a PC environment, for example, the register values and corresponding connections are:

- 0 (00H) through 15 (0FH) correspond to IRQ0 through IRQ15
- 16 (10H) through 254 (FEH) are reserved
- 255 (FFH) indicates "unknown" or "no connection"

The operating system or device driver can examine each device's interrupt pin and interrupt line register to determine which system interrupt request line the device uses to issue requests for service.

Table 47. ATU Interrupt Line Register - ATUILR

Register Offset +03CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	FFH	Interrupt Assigned - system-assigned value identifies which system interrupt controller's interrupt request line connects to the device's PCI interrupt request lines (as specified in the interrupt pin register). A value of FFH signifies "no connection" or "unknown".



2.13.25 ATU Interrupt Pin Register - ATUIPR

ATU Interrupt Pin Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the interrupt pin the ATU and Messaging Unit interface uses. The 81341 and 81342 is, a PCI single-function device and, as such, generates only one interrupt output. The interrupt output is for the Messaging Unit on **INTA#**.

Table 48. ATU Interrupt Pin Register - ATUIPR

Register Offset +03DH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	01H	Interrupt Used - A value of 01H signifies that the ATU interface unit uses INTA# as the interrupt pin.

2.13.26 ATU Minimum Grant Register - ATUMGNT

ATU Minimum Grant Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register specifies the burst period the device requires in increments of 8 PCI clocks.

This register and the ATU Maximum Latency register are information-only registers which the configuration uses to determine how often a bus master typically requires access to the PCI bus and the duration of a typical transfer when it does acquire the bus. This information is useful in determining the values to be programmed into the bus master latency timers and in programming the algorithm to be used by the PCI bus arbiter.

Table 49. ATU Minimum Grant Register - ATUMGNT

Register Offset +003EH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	80H	This register specifies how long a burst period the device needs in increments of 8 PCI clocks.



2.13.27 ATU Maximum Latency Register - ATUMLAT

ATU Maximum Latency Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register specifies how often the device needs to access the PCI bus in increments of 8 PCI clocks.

This register and the Minimum Grant Register are information-only registers which the configuration uses to determine how often a bus master typically requires access to the PCI bus and the duration of a typical transfer when it does acquire the bus. This information is useful in determining the values to be programmed into the bus master latency timers and in programming the algorithm to be used by the PCI bus arbiter.

Table 50. ATU Maximum Latency Register - ATUMLAT

Register Offset +03FH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	Specifies frequency (how often) the device needs to access the PCI bus in increments of 8 PCI clocks. A zero value indicates the device has no stringent requirement.



Inbound address translation for memory window 0 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The inbound translation base address for inbound window 0 is specified in [Section 2.13.13](#). When determining block size requirements — as described in [Section 2.13.23](#) — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in [Section 2.2.1.1](#).

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR0 have a direct effect on the IABAR0 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR0 makes the corresponding bit within the IABAR0 a read only bit which always returns 0. A value of 1 in a bit within the IALR0 makes the corresponding bit within the IABAR0 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR0, all writes to the IABAR0 has no effect since a value of all zeros within the IALR0 makes the IABAR0 a read only register.

Note: Bit 0 can be used to disable claiming of Memory Cycles that hit Inbound Memory Window 0 even though the host processor has allocated memory of the size requested by IABAR0/IALR0[31:12].

Table 51. Inbound ATU Limit Register 0 - IALRO

Intel® 81341 and 81342 I/O Processors
Developer's Manual
174



2.13.29 Inbound ATU Translate Value Register 0 - IATVR0

The Inbound ATU Translate Value Register 0 (IATVR0) in conjunction with the “[Inbound ATU Upper Translate Value Register 0 - IAUTVR0](#)” on [page 175](#) contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 52. Inbound ATU Translate Value Register 0 - IATVR0

Register Offset +044H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:12	FF000H	Inbound ATU Translation Value 0 - This value represents bits 31 to 12 of the internal bus address used to convert the PCI address to internal bus addresses. This value must be naturally aligned with the IABAR0 register's programmed value (see Section 2.13.23, "Determining Block Sizes for Base Address Registers" on page 169). The default address allows the ATU to access the internal 81341 and 81342 memory-mapped registers.
11:01	000H	Reserved
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read/write transactions through BAR0. When clear, no swap is performed. Refer to Section 2.3, "Big Endian Byte Swapping" on page 83 for more details.

2.13.30 Inbound ATU Upper Translate Value Register 0 - IAUTVR0

The Inbound ATU Upper Translate Value Register 0 (IAUTVR0) in conjunction with the "Inbound ATU Translate Value Register 0 - IATVR0" on page 175 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 53. Inbound ATU Upper Translate Value Register 0 - IAUTVR0

Register Offset +048H

Attribute Legend:

- RV = Reserved
- RC = Read Clear
- PR = Preserved
- RS = Read/Set
- RW = Read/Write
- RO = Read Only
- NA = Not Accessible



Inbound address translation for memory window 1 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The 81341 and 81342 value register programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR1 have a direct effect on the IABAR1 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR1 makes the corresponding bit within the IABAR1 a read only bit which always returns 0. A value of 1 in a bit within the IALR1 makes the corresponding bit within the IABAR1 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR1, all writes to the IABAR1 has no effect since a value of all zeros within the IALR1 makes the IABAR1 a read only register.

Note: Bit 0 can be used to disable claiming of Memory Cycles that hit Inbound Memory Window 1 even though the host processor has allocated memory of the size requested by IABAR1/IALR1[31:12].

The diagram illustrates the bit fields of the Inbound Memory Window 1 Claim Disable Register (04CH). The register is 32 bits wide, divided into two main sections: IOP Attributes (bits 31:12) and PCI Attributes (bits 11:0). The IOP Attributes section contains 20 bits, with the first 12 bits (31:20) being Read/Write (RW) and the remaining 8 bits (19:12) being Read/Write (RW) or Read/Write (RV). The PCI Attributes section contains 20 bits, with the first 12 bits (31:20) being Read/Write (RW) and the remaining 8 bits (19:12) being Read/Write (RW) or Read/Write (RV). The bits are grouped into four 5-bit fields, each with a blue vertical line indicating a boundary. The bit numbers 31, 28, 24, 20, 16, 12, 8, and 0 are labeled at the top. The attribute legend indicates: RW = Read/Write, RV = Reserved, RC = Read Clear, RO = Read Only, and NA = Not Accessible.

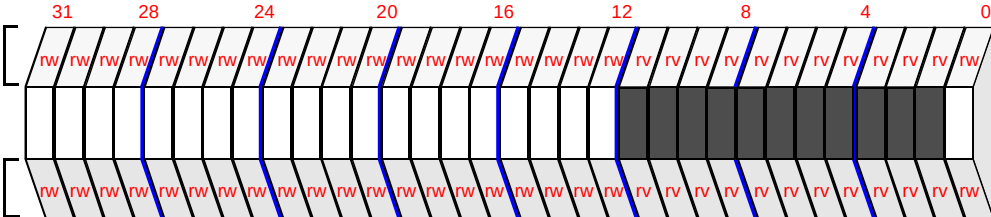
Bit	Default	Description
31:12	00000H	Inbound Translation Limit 1 - This value determines the memory block size required for the ATUS memory window 1.
11:01	000H	Reserved
00	0 ₂	Memory Window 1 Claim Disable -- When this bit is set, Inbound Memory Window 1 does not claim memory cycles on the PCI Bus. When clear, Inbound Memory Window 1 claims PCI Memory Cycles Normally.



2.13.32 Inbound ATU Translate Value Register 1 - IATVR1

The Inbound ATU Translate Value Register 1 (IATVR1) in conjunction with the “Inbound ATU Upper Translate Value Register 1 - IAUTVR1” on page 177 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the Inbound ATU address translation.

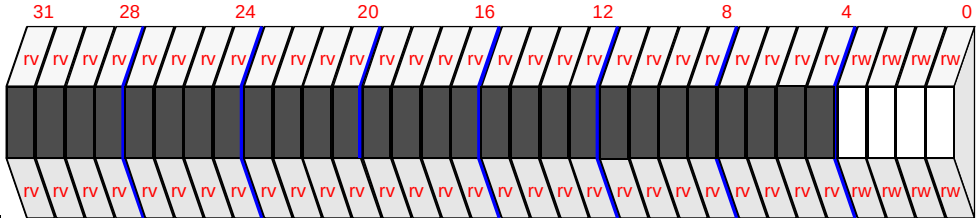
Table 55. Inbound ATU Translate Value Register 1 - IATVR1

																															
Register Offset +050H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																													
Bit	Default	Description																													
31:12	00000H	Inbound ATU Translation Value 1: Bits 31 to 12 of the internal bus address used to convert the PCI address to internal bus addresses. Must be naturally aligned with the IABAR1 register's programmed value (see Section 2.13.23, "Determining Block Sizes for Base Address Registers" on page 169).																													
11:01	000H	Reserved																													
00	0	Big Endian Byte Swap enable: 0 = No swap performed. See Section 2.3, "Big Endian Byte Swapping" on page 83 for more details. 1 = Performs a byte swap on all PCI read/write transactions through BAR1.																													

2.13.33 Inbound ATU Upper Translate Value Register 1 - IAUTVR1

The Inbound ATU Upper Translate Value Register 1 (IAUTVR1) in conjunction with the “Inbound ATU Translate Value Register 1 - IATVR1” on page 177 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 56. Inbound ATU Upper Translate Value Register 1 - IAUTVR1

			
Register Offset +054H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description	
31:04	000 0000H	Reserved	
3:0	0H	Inbound Upper ATU Translation Value 1 - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.	



2.13.34 Inbound ATU Limit Register 2 - IALR2

Inbound address translation for inbound window 2 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

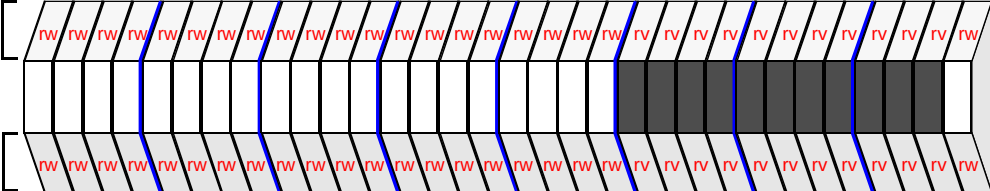
The inbound translation base address for inbound window 2 is specified in [Section 2.13.17](#). When determining block size requirements — as described in [Section 2.13.23](#) — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in [Section 2.2.1.1](#).

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR2 have a direct effect on the IABAR2 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR2 makes the corresponding bit within the IABAR2 a read only bit which always returns 0. A value of 1 in a bit within the IALR2 makes the corresponding bit within the IABAR2 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR2, all writes to the IABAR2 has no effect since a value of all zeros within the IALR2 makes the IABAR2 a read only register.

Note: Bit 0 can be used to disable claiming of PCI Cycles that hit Inbound Window 1 even though the host processor has allocated memory of the size requested by IABAR2/IALR2[31:12].

Table 57. Inbound ATU Limit Register 2 - IALR2

Register Offset +058H																																	
		Attribute Legend:														RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RO = Read Only NA = Not Accessible																	
Bit	Default	Description																															
31:12	00000H	Inbound Translation Limit 2 - This value determines the memory block size required for the ATUs memory window 2.																															
11:01	000H	Reserved																															
00	0 ₂	Window 2 Claim Disable -- When this bit is set, Inbound Window 2 does not claim cycles on the PCI Bus when clear, Inbound Memory Window 2 claims PCI Cycles Normally.																															



2.13.35 Inbound ATU Translate Value Register 2 - IATVR2

The Inbound ATU Translate Value Register 2 (IATVR2) in conjunction with the “Inbound ATU Upper Translate Value Register 2 - IAUTVR2” on page 179 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the Inbound ATU address translation.

Table 58. Inbound ATU Translate Value Register 2 - IATVR2

Register Offset +05CH		PCI Configuration Address Offset 5CH - 5FH	
		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description	
31:12	00000H	Inbound ATU Translation Value 2 - This value represents bits 31 to 12 of the internal bus address used to convert the PCI address to internal bus addresses. This value must be naturally aligned with the IABAR2 register's programmed value (see Section 2.13.23, “Determining Block Sizes for Base Address Registers” on page 169).	
11:01	000H	Reserved	
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read/write transactions through BAR2. When clear, no swap is performed. Refer to Section 2.3, “Big Endian Byte Swapping” on page 83 for more details.	

2.13.36 Inbound ATU Upper Translate Value Register 2 - IAUTVR2

The Inbound ATU Upper Translate Value Register 2 (IAUTVR2) in conjunction with the “Inbound ATU Translate Value Register 2 - IATVR2” on page 179 contain bits 35 to 8 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 59. Inbound ATU Upper Translate Value Register 2 - IAUTVR2

Register Offset +060H		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description	
31:04	000 0000H	Reserved	
3:0	0H	Inbound Upper ATU Translation Value 2 - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.	

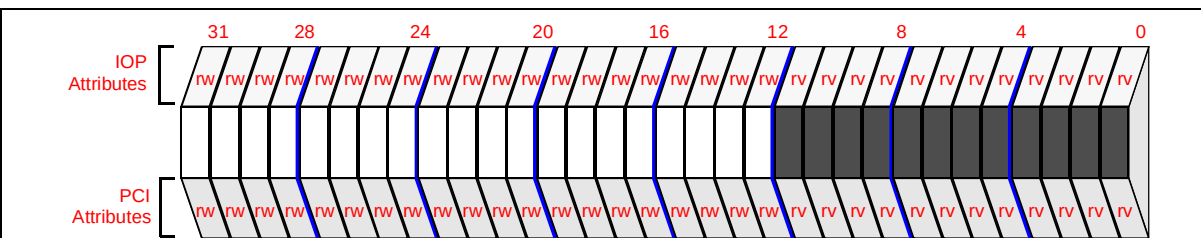


2.13.37 Expansion ROM Limit Register - ERLR

The Expansion ROM Limit Register (ERLR) defines the block size of addresses the ATU defines as Expansion ROM address space. Block size is programmed by writing a value into the ERLR.

Bits 31 to 12 within the ERLR have a direct effect on the ERBAR register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the ERLR makes the corresponding bit within the ERBAR a read only bit which always returns 0. A value of 1 in a bit within the ERLR makes the corresponding bit within the ERBAR read/write from PCI.

Table 60. Expansion ROM Limit Register - ERLR

					
Register Offset +064H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set		RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description			
31:12	000000H	Expansion ROM Limit - Memory block size required for Expansion ROM translation unit. Default value 0, indicates no Expansion ROM address space and all bits within ERBAR are read only with a value of 0.			
11:00	000H	Reserved			



2.13.38 Expansion ROM Translate Value Register - ERTVR

The Expansion ROM Translate Value Register 0 (ERTVR) in conjunction with the “Expansion ROM Upper Translate Value Register - ERUTVR” on page 181 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the Expansion ROM address translation.

Table 61. Expansion ROM Translate Value Register - ERTVR

		Register Offset +068H																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible															
Bit	Default	Description																															
31:12	00000H	Expansion ROM Translation Value - Bits 31 to 12 of internal bus address used to convert PCI address to internal bus addresses. This value must be naturally aligned with the ERBAR register's programmed value (see Section 2.13.23, “Determining Block Sizes for Base Address Registers” on page 169).																															
11:01	000H	Reserved																															
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read transactions through EROM BAR. When clear, no swap is performed. Refer to Section 2.3, “Big Endian Byte Swapping” on page 83 for more details.																															

2.13.39 Expansion ROM Upper Translate Value Register - ERUTVR

The Expansion ROM Upper Translate Value Register (ERUTVR) in conjunction with the “Expansion ROM Translate Value Register - ERTVR” on page 181 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the Expansion ROM address translation.

Table 62. Expansion ROM Upper Translate Value Register - ERUTVR

Internal Bus Address +06CH Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:04	000 0000H	Reserved
3:0	0H	Expansion ROM Upper Translation Value - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.



2.13.40 ATU Configuration Register - ATUCR

The ATU Configuration Register controls the outbound address translation for address translation unit. It also contains bits for Conventional PCI Delayed Read Command (DRC) aliasing, discard timer status, **P_SERR#** manual assertion, **P_SERR#** detection interrupt masking, and ATU BIST interrupt enabling.

Table 63. ATU Configuration Register - ATUCR

Register Offset +070H		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible
Bit	Default	Description
31:20	00H	Reserved
19	0 ₂	ATU DRC Alias - when set, the ATU does not distinguish read commands when attempting to match a current PCI read transaction with read data enqueued within the DRC buffer. When clear, a current read transaction must have the exact same read command as the DRR for the ATU to deliver DRC data. Not applicable in the PCI-X mode.
18:17	00b	Reserved
16	0 ₂	P_SERR# Manual Assertion - when set, the ATU asserts P_SERR# for one clock on the PCI interface. Until cleared, P_SERR# may not be manually asserted again. Once cleared, operation proceeds as specified.
15	0 ₂	ATU Discard Timer Status - when set, one of the 4 discard timers within the ATU has expired and discarded the delayed completion transaction within the queue. When clear, no timer has expired.
14:10	00 ₂	Reserved
09	0 ₂	P_SERR# Detected Interrupt Enable - When set, the Intel XScale® processor is signalled an HPI# interrupt when the ATU detects that P_SERR# was asserted. When clear, the Intel XScale® processor is not interrupted when P_SERR# is detected.
08	0 ₂	Halt ATU On Error Enable - When set and a Data Parity, Master-Absort, or Target-Absort error occurs on an Outbound PCI Write Request, the ATU halts operation using the following sequence: 1. Stop accepting Outbound Requests by clearing the Outbound ATU Enable (bit 1 of the ATUCR). 2. Flush any pending outbound write requests from the transaction queue. 3. Pull the Intel XScale® processor interrupt as normal for the error. Firmware should wait for the Outbound Queue Busy Bit in "PCI Configuration and Status Register - PCSR" to clear before setting the Outbound ATU Enable (ATUCR[1]). Note: Due to the pipelined nature of PERR# signaling, one additional write may have issued on the PCI bus before the error condition is detected. All remaining writes in the queue is flushed. Read Requests are allowed to proceed and complete normally though no new ones are accepted.
07:04	000 ₂	Reserved
03	0 ₂	ATU BIST Interrupt Enable - When set, enables an interrupt to the Intel XScale® processor when the start BIST bit is set in the ATUBISTR register. This bit is also reflected as BIST Capable bit 7 in the ATUBISTR register.
02	0 ₂	Reserved
01	0 ₂	Outbound ATU Enable - When set, enables the outbound address translation unit. When cleared, disables the outbound ATU.
00	0 ₂	Reserved



2.13.41 PCI Configuration and Status Register - PCSR

The PCI Configuration and Status Register has additional bits for controlling and monitoring various features of the PCI bus interface.

Table 64. PCI Configuration and Status Register - PCSR (Sheet 1 of 3)

31 28 24 20 16 12 8 4 0 IOP Attributes rw rw ro ro ro ro ro ro ro ro rw rc rw rw rw ro ro rv rv ro ro ro rc rv rv rv rv ro co co PCI Attributes rw rw ro ro ro ro ro ro ro ro rw rc rw rw rw rw ro ro rv rv ro ro ro rc rv rv rv rv ro co co		Register Offset +074H Attribute Legend: RW = Read/Write RV = Reserved CO = Clear Only RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:30	0 ₂	Initiate Core Processor Reset. These bits are used to place the Intel XScale® processor processors into reset. When a '1' is written to this bits the corresponding Core Processor Reset (Bit 1:0) is set. This bit is self clearing and always reads '0'.
29	HS_LSTAT	Hot-Swap Latch Status: Reflects the state of the HS_LSTAT pin.
28	HS_SM#	Hot-Swap Start up Mode: Indicates 81341 and 81342 behavior when targeted by Configuration transactions with Configuration Retry Mode enabled (bit 2 of the PCSR). 0 = 81341 and 81342 ignores all configuration transactions targeting it thereby causing the transaction to ultimately Master Abort. 1 = 81341 and 81342 retries all configuration transactions targeting it.
27:26	HS_FREQ[1:0]	Hot-Swap Frequency selection These bits reflect the value on the HS_FREQ[1:0] pins when the HS_SM# strap is asserted. 11 PCI Mode, 33 or 66 MHz. Use P_M66EN to determine frequency 10 PCI-X Mode, 66 MHz 01 PCI-X Mode, 100 MHz 00 PCI-X Mode, 133 MHz Note: 81341 and 81342 does not support CompactPCI Hot-Swap in PCI-X Mode2.
25	PCIX_EP#	PCIX End Point Strap 0 = ATUX is operating in End Point mode 1 = ATUX is operating as a central resource.
24	PCIXM1_100#	PCIX Mode1 - 100MHz limit When all bus devices are capable of operating at greater then 66MHz, the PCIXM1_100# strap limits the bus to operate at a maximum frequency of 100MHz. 0 = PCI-X Bus is limited to 100MHz operation in Mode 1 1 = PCI-X Bus can operate at 133MHz. in Mode 1
23	PCIXM2_100#	PCIX Mode2 - 100MHz limit (200MHz data rate) When all bus devices are capable of operating at greater then 66MHz, the PCIXM1_100# strap limits the bus to operate at a maximum frequency of 100MHz. 0 = PCI-X Bus is limited to 100MHz operation in Mode 1 1 = PCI-X Bus can operate at 133MHz. in Mode 1
22	EXTARB#	External Arbiter When operating in Central Resource Mode (PCIX_EP# = 1), this strap enables the use of an external arbiter 0 = External Arbiter Enabled 1 = External Arbiter Disabled (Use Internal Arbiter).



Table 64. PCI Configuration and Status Register - PCSR (Sheet 2 of 3)

Register Offset +074H		Attribute Legend: RW = Read/Write RV = Reserved CO = Clear Only RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
21	1	Central Resource PCI Bus Reset - When set, P_RSTOUT# is asserted. When cleared, P_RSTOUT# is deasserted. After clearing this bit, there is a delay of about 300 uS before hardware de-asserts the P_RSTOUT# signal. After this bit is cleared, the hardware waits about 150 uS to allow the PLL to warm-up and another 150 uS to allow the clocks to stabilize. Therefore, firmware has to wait about 300uS for the P_RSTOUT# signal to get de-asserted. After hardware de-asserts P_RSTOUT#, firmware has to wait before issuing the first configuration cycle in order to meet the PCI timing parameter Trhfa (about 2^{26} PCI clocks). Note that the PCI timing parameter Trhfa is dependent on the PCI bus speed selected. Note: P_RSTOUT# is asserted by default. This output should remain unconnected when operating as an endpoint.
20	0 ₂	Detected Uncorrectable Address or Attribute Error - set when an uncorrectable error is detected during either the address or attribute phase of a transaction on the PCI bus even when the ATUCMD register's Parity Error Response bit is cleared. Set under the following conditions: Any Uncorrectable Address or Attribute (PCI-X Only) Error on the Bus (including one generated by the ATU).
19:16	See description for default value	PCI-X capability - These bits define the mode of the PCI bus (conventional or PCI-X) as well as the operating frequency in the case of PCI-X mode and are consistent with the electrical value on the PCI bus. As a Central Resource, this field controls the initialization pattern driven on the PCI bus during reset and the value driven on the CR_FREQ[1:0] pins. The default value of this field is dependent on the following pins/straps: PCIXM1_100#, PCIXM2_100#, P_MODE2, P_PCIXCAP, and P_M66EN. As an endpoint, this register reflects the value captured off the bus during reset based on the following signals: P_PERR#, P_DEVSEL#, P_STOP#, and P_TRDY#. 1111 - Conventional PCI mode(frequency depends on P_M66EN) 1110 - PCI-X 66 1101 - PCI-X 100 1100 - PCI-X 133 0110 - PCI-X 266 (66 MHz P_CLK) 0101 - PCI-X 266 (100 MHz P_CLK) 0100 - PCI-X 266 (133 MHz P_CLK) All other values are reserved. All other patterns are reserved or not supported by the 81341 and 81342. See Section 2.11.7, "Bus Mode and Frequency Initialization" on page 137 for more details.
15	0 ₂	Outbound Transaction Queue Busy: 0 = Outbound Transaction Queue Empty 1 = Outbound Transaction Queue Busy
14	0 ₂	Inbound Transaction Queue Busy: 0 = Inbound Transaction Queue Empty 1 = Inbound Transaction Queue Busy
13	0 ₂	Reserved
12	0 ₂	Discard Timer Value - This bit controls the time-out value for the four discard timers attached to the queues holding read data. A value of 0 indicates the time-out value is 2^{15} clocks. A value of 1 indicates the time-out value is 2^{10} clocks.
11	0 ₂	Reserved



Table 64. PCI Configuration and Status Register - PCSR (Sheet 3 of 3)

Register Offset +074H																		Attribute Legend: RW = Read/Write RV = Reserved CO = Clear Only RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible															
Bit	Default	Description																															
10	Varies with external state of P_M66EN at PCI bus reset	Conventional PCI Bus Operating at 66 MHz - When set, the interface has been initialized to function at 66 MHz in Conventional PCI mode by the assertion of P_M66EN during bus initialization. When clear, the interface has been initialized as a 33 MHz bus. Note: When PCSR bits 19:16 are not equal to zero, then this bit is meaningless since the 81341 and 81342 is operating in PCI-X mode.																															
09	PMODE2	PCI Bus Mode 2 Indicates the bus is capable of operating in PCI-X Mode 2.																															
08	Varies with external state of P_REQ64# at PCI bus reset	PCI Bus 64-Bit Capable - When clear, the PCI bus interface has been configured as 64-bit capable by the assertion of P_REQ64# on the rising edge of P_RST# . When set, the PCI interface is configured as 32-bit only. Note that this bit is valid only when acting as an endpoint.																															
07	Varies with external state of the FW_TIMER_OFF# strap	Firmware Timer Disable 0 = Firmware Timeout is disabled. 1 = Firmware Timeout is enabled When enabled, a 400mS timer is started at the trailing edge of reset. The Configuration Request Retry bit (bit 2 of this register) is cleared when it is not cleared before the expiration of the timer. For example, when firmware does not clear the Configuration Request Retry bit before the timer expires. The firmware timer bit is also automatically cleared by the firmware timer when the Configuration Request Retry bit is still set when the timer expires. After the host is allowed access, the firmware timer bit can be used to indicate how the Configuration Request Retry bit was cleared: 0 = The firmware timer expired and cleared both the Configuration Request Retry and the firmware timer bits. 1 = Firmware cleared the Configuration Request Retry bit before the timer expired. Note: When the firmware timer is disabled, firmware is responsible to clear the Configuration Request Retry bit. Otherwise, the ATU indefinitely retries all host configuration cycles.																															
06:03	00 ₂	Reserved																															
02	Varies with external state of RETRY pin at PCI bus reset	Configuration Cycle Retry - When this bit is set, the PCI interface of the 81341 and 81342 responds to all configuration cycles with a Retry condition. When clear, the 81341 and 81342 responds to the appropriate configuration cycles. The default condition for this bit is based on the external state of the RETRY pin at the rising edge of P_RST# . When the external state of the pin is high, the bit is set. When the external state of the pin is low, the bit is cleared.																															
01:00	Varies with external state of the RST_MODE[1:0]# pins at PCI bus reset	Core Processor Reset - These bits are set to their default values by the hardware when either P_RST# is asserted or the Reset Internal Bus bit in PCSR is set. When these bits are set, the associated Intel XScale® processors are being held in reset. Software cannot these bits. Software is required to clear these bit(s) to deassert Intel XScale® processor reset. The default condition for these bits are based on the external state of the RST_MODE[1:0]# pins at the rising edge of P_RST# . When the external state of these pins are low, the default value of these bits is set. When the external state of the pin is high, the default value of these bits is clear. Note: RST_MODE0# is associated with Intel XScale® processor 0 which is the Protocol Core while RST_MODE1# is associated with Intel XScale® processor 1 which is the Application core.																															



2.13.42 ATU Interrupt Status Register - ATUISR

The ATU Interrupt Status Register is used to notify the core processor of the source of an ATU interrupt. In addition, this register is written to clear the source of the interrupt to the interrupt unit of the 81341 and 81342. All bits in this register are Read/Clear.

Bits 4:0 are a direct reflection of bits 14:11 and bit 8 (respectively) of the ATU Status Register (these bits are set at the same time by hardware but need to be cleared independently). Bit 7 is set by an error associated with the internal bus of the 81341 and 81342. Bit 8 is for software BIST. The conditions that result in an ATU interrupt are cleared by writing a 1 to the appropriate bits in this register.

Note that bits 4:0, and bits 15 and 13:7 can result in an interrupt being driven to the Intel XScale® processor.

Table 65. ATU Interrupt Status Register - ATUISR (Sheet 1 of 2)

Register Offset +078H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:19	0000H	Reserved
18	0 ₂	PCI Interface Error - This bit is set when any PCI interface error occurs. When set, this bit results in the assertion of the ATU PCI Interface Error Interrupt.
17	0 ₂	VPD Address Register Updated - This bit is set when a PCI bus configuration write occurs to the VPDAR register. When set, this bit results in the assertion of the ATU Config Reg Write Interrupt.
16	0 ₂	Internal Bus Parity Error Detected - This bit is set when an Internal Bus Parity Error is detected.
15	0 ₂	ATU Configuration Write - This bit is set when a PCI bus configuration write occurs to any enabled function. When set, this bit results in the assertion of the ATU Config Reg Write Interrupt.
14	0 ₂	Detected Correctable Error - This bit is set in PCI-X Mode 2 only when the 81341 and 81342 detects a single bit ECC error in any phase of a PCI transaction.
13	0 ₂	Initiated Split Completion Error Message - This bit is set when the device initiates a Split Completion Message on the PCI Bus with the Split Completion Error attribute bit set.
12	0 ₂	Received Split Completion Error Message - This bit is set when the device receives a Split Completion Message from the PCI Bus with the Split Completion Error attribute bit set.
11	0 ₂	Power State Transition - When the Power State Field of the ATU Power Management Control/Status Register is written to transition the ATU function Power State from D0 to D3, D0 to D1, or D3 to D0 and the ATU Power State Transition Interrupt mask bit is cleared, this bit is set.
10	0 ₂	P_SERR# Asserted - set when P_SERR# is asserted on the PCI bus by the ATU.
09	0 ₂	Detected Parity Error - set when a parity error is detected on the PCI bus even when the ATUCMD register's Parity Error Response bit is cleared. Set under the following conditions: - Uncorrectable Write Data Error when the ATU is a target (inbound write). - Uncorrectable Read Data Error when the ATU is an initiator (outbound read). - Any Uncorrectable Address or Attribute (PCI-X Only) Error on the Bus.
08	0 ₂	ATU BIST Interrupt - When set, the host processor has set the start BIST, ATUBISTR register bit 6, and the ATU BIST interrupt enable (ATUCR register bit 3) is enabled. The Intel XScale® processor can initiate the software BIST and store the result in ATUBISTR register bits 3:0.

Table 65. ATU Interrupt Status Register - ATUISR (Sheet 2 of 2)

Register Offset
+078H

Attribute Legend:
 RV = Read/Write
 RC = Read Clear
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



The ATU Interrupt Mask Register contains the control bit to enable and disable interrupts generated by the ATU.

Bit	Default	Description
31:16	0 0000H	Reserved
15	0 ₂	PCI Interface Error Mask - Controls the setting of bit 18 of the ATUISR and generation of the PCI Interface error when any PCI bus error occurs. 0 = Not Masked 1 = Masked
14	0 ₂	VPD Address Register Updated Mask - Controls the setting of bit 17 of the ATUISR and generation of the ATU Configuration Register Write interrupt when a PCI bus write occurs to the VPDAR register. 0 = Not Masked 1 = Masked
13	0 ₂	Internal Bus Parity Error Detected Mask - Controls the setting of bit 16 of the ATUISR and generation of the ATU Error interrupt when an internal bus parity error is detected. 0 = Not Masked 1 = Masked
12	0 ₂	Configuration Register Write Mask - Controls the setting of bit 15 of the ATUISR and generation of the ATU Configuration Register Write interrupt when a PCI bus write occurs to any ATU configuration register except those covered by mask bit 11 and bit 14 of this register. 0 = Not Masked 1 = Masked
11	1 ₂	Detected Correctable Error Mask - Controls setting of bit 14 of the ATUISR and generation of the Correctable Error interrupt when in PCI-X Mode 2, a correctable error is detected in any phase of a PCI transaction. 0 = Not Masked 1 = Masked
10	0 ₂	Initiated Split Completion Error Message Interrupt Mask - Controls the setting of bit 13 of the ATUISR and generation of the ATU Error interrupt when the ATU initiates a Split Completion Error Message. 0 = Not Masked 1 = Masked
09	0 ₂	Received Split Completion Error Message Interrupt Mask- Controls setting of bit 12 of ATUISR and generation of ATU Error interrupt when a Split Completion Error Message results in bit 29 of PCIXSR being set. 0 = Not Masked 1 = Masked
08	1 ₂	Power State Transition Interrupt Mask - Controls the setting of bit 12 of the ATUISR and generation of the ATU Error interrupt when ATU Power Management Control/Status Register is written to transition the ATU Function Power State from D0 to D3, D0 to D1, D1 to D3 or D3 to D0. 0 = Not Masked 1 = Masked
07	0 ₂	ATU Detected Parity Error Interrupt Mask - Controls the setting of bit 9 of the ATUISR and generation of the ATU Error interrupt when a parity error detected on the PCI bus that sets bit 15 of the ATUSR. 0 = Not Masked 1 = Masked

**Table 66. ATU Interrupt Mask Register - ATUIMR (Sheet 2 of 2)**

IOP Attributes PCI Attributes		
Register Offset +07CH		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
06	0 ₂	ATU SERR# Asserted Interrupt Mask - Controls the setting of bit 10 of the ATUISR and generation of the ATU Error interrupt when SERR# is asserted on the PCI interface resulting in bit 14 of the ATUSR being set. 0 = Not Masked 1 = Masked Note: This bit is specific to the ATU asserting SERR# and not detecting SERR# from another master.
05	0 ₂	ATU PCI Master Abort Interrupt Mask - Controls the setting of bit 3 of the ATUISR and generation of the ATU Error interrupt when a master abort error resulting in bit 13 of the ATUSR being set. 0 = Not Masked 1 = Masked
04	0 ₂	ATU PCI Target Abort (Master) Interrupt Mask- Controls the setting of bit 12 of the ATUISR and ATU Error generation of the interrupt when a target abort error resulting in bit 12 of the ATUSR being set 0 = Not Masked 1 = Masked
03	0 ₂	ATU PCI Target Abort (Target) Interrupt Mask- Controls the setting of bit 1 of the ATUISR and generation of the ATU Error interrupt when a target abort error resulting in bit 11 of the ATUSR being set. 0 = Not Masked 1 = Masked
02	0 ₂	ATU PCI Master Parity Error Interrupt Mask - Controls the setting of bit 0 of the ATUISR and generation of the ATU Error interrupt when a parity error resulting in bit 8 of the ATUSR being set. 0 = Not Masked 1 = Masked
01	0 ₂	ATU Inbound Error SERR# Enable - Controls when ATU asserts (when enabled through the ATUCMD) SERR# on the PCI interface in response to a master abort on the internal bus during an inbound write transaction. 0 = SERR# Not Asserted due to error 1 = SERR# Asserted due to error
00	0 ₂	ATU ECC Target Abort Enable - Controls the ATU response on the PCI interface to a target abort (ECC error) from the memory controller on the internal bus. In conventional mode, this action only occurs during an inbound read transaction where the data phase that was target aborted on the internal bus is actually requested from the inbound read queue. 0 = Disconnect with data (the data being up to 64 bits of 1's) 1 = Target Abort Note: In PCI-X Mode, The ATU initiates a Split Completion Error Message (with message class=2h - completer error and message index=81h - 81341 and 81342 internal bus target abort) on the PCI bus, independent of the setting of this bit.



2.13.44 VPD Capability Identifier Register - VPD_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the VPD extended capability with an ID of 03H as defined by the *PCI Local Bus Specification*, Revision 2.3.

Table 67. VPD Capability Identifier Register - VPD_Cap_ID

Register Offset +090H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	03H	Cap_Id - This field with its 03H value identifies this item in the linked list of Extended Capability Headers as being the VPD capability registers.

2.13.45 VPD Next Item Pointer Register - VPD_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 81341 and 81342, this the final capability list, and hence, this register is set to 00H.

Table 68. VPD Next Item Pointer Register - VPD_Next_Item_Ptr

Register Offset +091H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list. Since the VPD capabilities are the last in the linked list of extended capabilities in the 81341 and 81342, the register is set to 00H.



2.13.46 VPD Address Register - VPDAR

The VPD Address register (VPDAR) contains the DWORD-aligned byte address of the VPD to be accessed. The register is read/write and the initial value at power-up is indeterminate.

A PCI Configuration Write to the VPDAR interrupts the Intel XScale® processor. Software can use the Flag setting to determine whether the configuration write was intended to initiate a read or write of the VPD through the VPD Data Register.

Table 69. VPD Address Register - VPDAR

Register Offset +092H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15	0 ₂	Flag - A flag is used to indicate when a transfer of data between the VPD Data Register and the storage component has completed. Please see Section 2.10, "Vital Product Data" on page 132 for more details on how the 81341 and 81342 handles the data transfer.
14:0	0000H	VPD Address - This register is written to set the DWORD-aligned byte address used to read or write Vital Product Data from the VPD storage component.

2.13.47 VPD Data Register - VPDDR

This register is used to transfer data between the 81341 and 81342 and the VPD storage component.

Table 70. VPD Data Register - VPDDR

Register Offset +094H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:0	0000H	VPD Data - Four bytes are always read or written through this register to/from the VPD storage component.



2.13.48 PM Capability Identifier Register - PM_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the PCI Bus Power Management extended capability with an ID of 01H as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 71. PM_Capability Identifier Register - PM_Cap_ID

IOP Attributes PCI Attributes		
Register Offset +098H	PCI Configuration Offset 98H	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
07:00	01H	Cap_Id - This field with its' 01H value identifies this item in the linked list of Extended Capability Headers as being the PCI Power Management Registers.

2.13.49 PM Next Item Pointer Register - PM_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 81341 and 81342, the next capability (MSI-X capability list) is located at off-set B0H. Note that the PM_Next_Item_Ptr can be written by the processor.

Table 72. PM Next Item Pointer Register - PM_Next_Item_Ptr

IOP Attributes PCI Attributes		
Register Offset +099H	PCI Configuration Offset 99H	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
07:00	B0H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list which in the 81341 and 81342 is the MSI-X extended capabilities header.



2.13.50 ATU Power Management Capabilities Register - APMCR

Power Management Capabilities bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This register is a 16-bit read-only register which provides information on the capabilities of the ATU function related to power management.

Table 73. ATU Power Management Capabilities Register - APMCR

Register Offset +09AH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:11	00000 ₂	PME_Support - This function is not capable of asserting the PME# signal in any state, since PME# is not supported by the 81341 and 81342.
10	0 ₂	D2_Support - This bit is set to 0 ₂ indicating that the 81341 and 81342 does not support the D2 Power Management State
9	1 ₂	D1_Support - This bit is set to 1 ₂ indicating that the 81341 and 81342 supports the D1 Power Management State
8:6	000 ₂	Aux_Current - This field is set to 000 ₂ indicating that the 81341 and 81342 has no current requirements for the 3.3Vaux signal as defined in the <i>PCI Bus Power Management Interface Specification</i> , Revision 1.1
5	0 ₂	DS1 - This field is set to 0 ₂ meaning that this function does not require a device specific initialization sequence following the transition to the D0 uninitialized state.
4	0 ₂	Reserved.
3	0 ₂	PME Clock - Since the 81341 and 81342 does not support PME# signal generation this bit is cleared to 0 ₂ .
2:0	010 ₂	Version - Setting these bits to 010 ₂ means that this function complies with <i>PCI Bus Power Management Interface Specification</i> , Revision 1.1



2.13.51 ATU Power Management Control/Status Register - APMCSR

Power Management Control/Status bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This 16-bit register is the control and status interface for the power management extended capability.

Table 74. ATU Power Management Control/Status Register - APMCSR

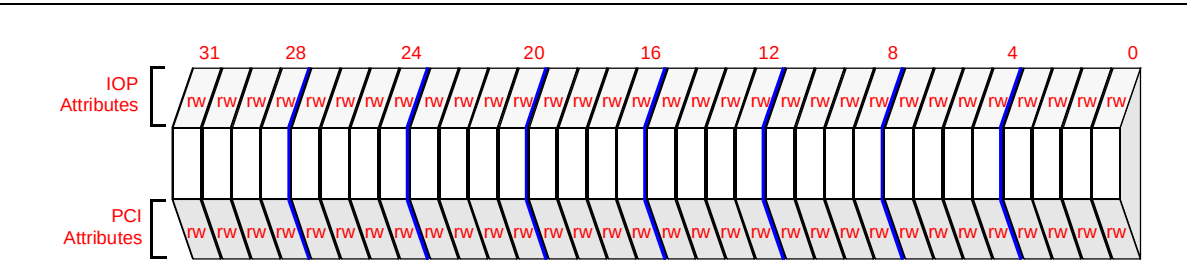
Register Offset +09CH				Attribute Legend: RW = Read/Write RC = Read Clear PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description			
15	0 ₂	PME_Status - This function is not capable of asserting the PME# signal in any state, since PME# is not supported by the 81341 and 81342.			
14:9	00H	Reserved			
8	0 ₂	PME_En - This bit is hard-wired to read-only 0 ₂ since this function does not support PME# generation from any power state.			
7:2	000000 ₂	Reserved			
1:0	00 ₂	Power State - This 2-bit field is used both to determine the current power state of a function and to set the function into a new power state. The definition of the values is: 00 ₂ - D0 01 ₂ - D1 10 ₂ - D2 (Unsupported) 11 ₂ - D3 _{hot} The 81341 and 81342 supports only the D0 and D3 _{hot} states.			



2.13.52 ATU Scratch Pad Register - ATUSPR

This register can be used for application specific purposes and has no direct impact on the hardware.

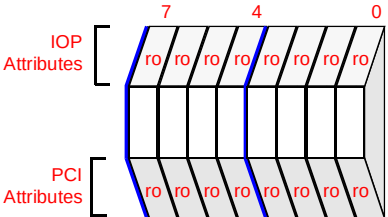
Table 75. Scratch Pad Register - ATUSPR

		
Internal Bus Address Offset +0CCH		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible
Bit	Default	Description
31:0	0000H	Scratch Pad Data - Entire register is available for application specific purposes.

2.13.53 PCI-X Capability Identifier Register - PCI-X_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the PCI-X extended capability with an ID of 07H as defined by the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0.

Table 76. PCI-X_Capability Identifier Register - PCI-X_Cap_ID

Register Offset +0D0H				PCI Configuration Offset D0H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description					
07:00	07H	Cap_Id - This field with its' 07H value identifies this item in the linked list of Extended Capability Headers as being the PCI-X capability registers.					



2.13.54 PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list.

Table 77. PCI-X Next Item Pointer Register - PCI-X_Next_Item_Ptr

Register Offset +0D1H PCI Configuration Offset D1H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	E8H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list which in the 81341 and 81342 is the CompactPCI extended capabilities header.

2.13.55 PCI-X Command Register - PCIXCMD

This register controls various modes and features of ATU and Message Unit when operating in the PCI-X mode.

Table 78. PCI-X Command Register - PCIXCMD (Sheet 1 of 2)

Register Offset +0D2H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:14	00 ₂	Reserved
13:12	01 ₂	PCI-X Capabilities List Item Version - This field indicates that the 81341 and 81342 implements version 1 of the PCI-X Capabilities list item defined in the <i>PCI-X Protocol Addendum to the PCI Local Bus Specification</i> , Revision 2.0. Specifically, the 81341 and 81342 implements the 24 byte format of the PCI-X capabilities list item and support for ECC in Mode 2 only.
11:7	00000 ₂	Reserved

**Table 78. PCI-X Command Register - PCIXCMD (Sheet 2 of 2)**

Register Offset +0D2H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
6:4	011 ₂	Maximum Outstanding Split Transactions - This register sets the maximum number of Split Transactions the device is permitted to have outstanding at one time. Register Maximum Outstanding 0 1 1 2 2 3 3 4 4 8 5 12 6 16 7 32
3:2	00 ₂	Maximum Memory Read Byte Count - This register sets the maximum byte count the device uses when initiating a Sequence with one of the burst memory read commands. Register Maximum Byte Count 0 512 1 1024 2 2048 3 4096
1	1 ₂	Enable Relaxed Ordering - When set, the 81341 and 81342 may set the relaxed ordering bit in the Requester Attributes of Transactions.
0	0 ₂	Uncorrectable Data Error Recovery Enable - The device driver sets this bit to enable the device to attempt to recover from uncorrectable data errors. When this bit is 0 and the device is in PCI-X mode, the device asserts P_SERR# (when enabled) whenever the Master Data Parity Error bit (Status register, bit 8) is set.



2.13.56 PCI-X Status Register - PCIXSR

This register identifies the capabilities and current operating mode of ATU when operating in the PCI-X mode.

Table 79. PCI-X Status Register - PCIXSR (Sheet 1 of 2)

Register Offset +0D4H																																	
		Attribute Legend: RV = Reserved RC = Read Clear RO = Read Only RS = Read/Set RW = Read/Write NA = Not Accessible																															
Bit	Default	Description																															
31:30	00 ₂	Reserved																															
29	0 ₂	Received Split Completion Error Message - This bit is set when the device receives a Split Completion Message with the Split Completion Error attribute bit set. Once set, this bit remains set until software writes a 1 to this location. 0 = no Split Completion error message received. 1 = a Split Completion error message has been received.																															
28:26	001 ₂	Designed Maximum Cumulative Read Size (DMCRS) - The value of this register depends on the setting of the Maximum Memory Read Byte Count field of the PCIXCMD register: DMCRS Max ADQs <i>Maximum Memory Read Byte Count Register Setting</i> 1 16 512 (Default) 2 32 1024 2 32 2048 2 32 4096																															
25:23	011 ₂	Designed Maximum Outstanding Split Transactions - The 81341 and 81342 can have up to four outstanding split transactions.																															
22:21	01 ₂	Designed Maximum Memory Read Byte Count - The 81341 and 81342 can generate memory reads with byte counts up to 1024 bytes.																															
20	1 ₂	81341 and 81342 is a complex device.																															
19	0 ₂	Unexpected Split Completion - This bit is set when an unexpected Split Completion with this device's Requester ID is received. Once set, this bit remains set until software writes a 1 to this location. 0 = no unexpected Split Completion has been received. 1 = an unexpected Split Completion has been received.																															
18	0 ₂	Split Completion Discarded - This bit is set when the device discards a Split Completion because the requester would not accept it. See Section 5.4.4 of the <i>PCI-X Protocol Addendum to the PCI Local Bus Specification</i> , Revision 2.0 for details. Once set, this bit remains set until software writes a 1 to this location. 0 = no Split Completion has been discarded. 1 = a Split Completion has been discarded. Note: The 81341 and 81342 never sets this bit since there is no Inbound address responding to Inbound Read Requests with Split Responses (Memory or Register) that has "read side effects."																															
17	1 ₂	81341 and 81342 is a 133 MHz capable device.																															
16	32BITPCI#	81341 and 81342 can be configured to identify the add-in card to the system as 64-bit or 32-bit wide via a user-configurable strap (32BITPCI#). This strap, by default, identifies the 81341 and 81342 subsystem as 64-bit unless the user attaches the appropriate pull-down resistor to the strap. 0 = The bus is 32 bits wide. 1 = The bus is 64 bits wide.																															

**Table 79. PCI-X Status Register - PCIXSR (Sheet 2 of 2)**

Register Offset +0D4H																																	
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set														RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																	
Bit	Default	Description																															
15:8	FFH	Bus Number - This register is read for diagnostic purposes only. It indicates the number of the bus segment for the device containing this function. The function uses this number as part of its Requester ID and Completer ID. For all devices other than the source bridge, each time the function is addressed by a Configuration Write transaction, the function must update this register with the contents of AD[7:0] of the attribute phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction when all of the following are true: 1. The transaction uses a Configuration Write command. 2. IDSEL is asserted during the address phase. 3. AD[1:0] are 00b (Type 0 configuration transaction). 4. AD[10:08] of the configuration address contain the appropriate function number.																															
7:3	1FH	Device Number - This register is read for diagnostic purposes only. It indicates the number of the device containing this function, i.e., the number in the Device Number field (AD[15:11]) of the address of a Type 0 configuration transaction that is assigned to the device containing this function by the connection of the system hardware. The system must assign a device number other than 00h (00h is reserved for the source bridge). The function uses this number as part of its Requester ID and Completer ID. Each time the function is addressed by a Configuration Write transaction, the device must update this register with the contents of AD[15:11] of the address phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction when all of the following are true: 1. The transaction uses a Configuration Write command. 2. IDSEL is asserted during the address phase. 3. AD[1:0] are 00b (Type 0 configuration transaction). 4. AD[10:08] of the configuration address contain the appropriate function number.																															
2:0	000 ₂	Function Number - This register is read for diagnostic purposes only. It indicates the number of this function; i.e., the number in the Function Number field (AD[10:08]) of the address of a Type 0 configuration transaction to which this function responds. The function uses this number as part of its Requester ID and Completer ID.																															



2.13.57 ECC Control and Status Register - ECCCSR

The ECCCSR register provides additional information about ECC errors that occurred on the PCI bus. Registers that store information from the failing transaction always store information directly from the PCI bus (uncorrected), even when correction of the error is possible.

Note: The “ECC Control and Status Register - ECCCSR”, “ECC First Address Register - ECCFAR”, “ECC Second Address Register - ECCSAR”, and “ECC Attribute Register - ECCAR” report the actual transaction that has the error. For example, when the Split Completion of an original Outbound Read request has an error, the information regarding the Split Completion is reported.

Table 80. ECC Control and Status Register - ECCCSR (Sheet 1 of 3)

Internal Bus Address +0D8H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only WO = Write Only NA = Not Accessible	
Bit	Default	Description	
31	0 ₂ (Mode 1 or Conventional) 1 ₂ (Mode 2)	ECC Mode - When this bit is 1, the 81341 and 81342 is in ECC mode. When this bit is 0, the 81341 and 81342 is in parity mode. The state of this bit after P_RST# is determined by the PCI-X initialization pattern described in the <i>PCI-X Protocol Addendum to the PCI Local Bus Specification</i> , Revision 2.0. In PCI-X Mode 2, this bit is a 1, otherwise this bit is a 0. Note: The 81341 and 81342 does not support ECC in PCI-X Mode 1 or in Conventional PCI mode.	
30	0 ₂	Disable Single-Bit-Error Correction - When the 81341 and 81342 is in ECC mode and this bit is 0, correctable errors (as described in the <i>PCI-X Protocol Addendum to the PCI Local Bus Specification</i> , Revision 2.0) are corrected. When the 81341 and 81342 is in ECC mode and this bit is 1, correctable errors are not corrected and are treated as uncorrectable errors, including the setting of status bits and assertion of error indicator signals on the bus. Disabling single-bit error correction enhances the error detection capability of the ECC. In parity mode (ECC Mode bit is 0), this bit has no meaning and is ignored by the 81341 and 81342. Note: Writes to this register do not affect this bit unless the ECC Control Update Enable bit is a 1 in the data pattern being written.	
29	0 ₂	Reserved	
28	0 ₂	ECC Control Update Enable - This bit always reads as a 0. When this bit is 1 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are also updated (written). When this bit is 0 in the data pattern being written, the Disable Single-Bit-Error Correction and ECC Mode bits are not updated.	
27:24	0H	Error Upper Attributes - When the ECC Error Phase register is non-zero, this register indicates the contents of the P_C/BE[3:0]# bus for the attribute phase of the transaction that included the error.	
23:20	0H	Error Second Command - When the ECC Error Phase register is non-zero and the transaction that included the error used a dual address cycle, this register indicates the contents of the P_C/BE[3:0]# bus for the second address phase of the transaction that included the error.	
19:16	0H	Error First (or only) Command - When the ECC Error Phase register is non-zero, this register indicates the contents of the P_C/BE[3:0]# bus for the first (or only) address phase of the transaction that included the error.	

**Table 80. ECC Control and Status Register - ECCCSR (Sheet 2 of 3)**

31 28 24 20 16 12 8 4 0 IOP Attributes ro rw rv wo ro</	



Internal Bus Address +0D8H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only WO = Write Only NA = Not Accessible		
Bit	Default	Description
2	0 ₂	Additional Correctable ECC Error - This bit is set when the 81341 and 81342 detects a correctable ECC error while error correction is enabled and the device is already indicating some other ECC error (i.e. the ECC Error Phase register is non-zero). 0 = No additional correctable ECC error has been detected. 1 = One or more additional correctable ECC errors have been detected.
1:0	00 ₂	Reserved



2.13.58 ECC First Address Register - ECCFAR

When the ECC Error Phase register (bits 6:4 of the ECCCSR) is non-zero (indicating that an error has been captured), the ECCFAR register indicates the contents of the **P_AD[31:0]** bus (for 64- and 32-bit buses) for the address phase of the transaction that included the error. For Dual Address Cycle (DAC) transactions, this represents the least significant 32-bits of the 64-bit address. When the ECC Error Phase register is zero, the contents of this register are undefined.

Note: Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even when correction of the error is possible.

Note: The "ECC Control and Status Register - ECCCSR", "ECC First Address Register - ECCFAR", "ECC Second Address Register - ECCSAR", and "ECC Attribute Register - ECCAR" report the actual transaction that has the error. For example, when the Split Completion of an original Outbound Read request has an error, the information regarding the Split Completion is reported.

Table 81. ECC First Address Register - ECCFAR

Internal Bus Address +0DCH		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible	
Bit	Default	Description	
31:00	0000 0000H	ECC First Address - This register represents the 32-bit address of a failing single address cycle (SAC) transaction. Or, in the case of a failing DAC transaction, this register represents the least significant 32 bits of the address.	



When the ECC Error Phase register (bits 6:4 of the ECCCSR) is non-zero (indicating that an error has been captured) and the failing transaction included a dual address cycle (DAC), the ECCSAR register indicates the contents of the **P_AD[31:0]** bus (for 64- and 32-bit buses) for the second address phase of the transaction that included the error. When the ECC Error Phase register is zero, the contents of this register are undefined.

Note: The “ECC Control and Status Register - ECCCSR”, “ECC First Address Register - ECCFAR”, “ECC Second Address Register - ECCSAR”, and “ECC Attribute Register - ECCAR” report the actual transaction that has the error. For example, when the Split Completion of an original Outbound Read request has an error, the information regarding the Split Completion is reported.

Register Offset +0E0H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	0000 0000H	ECC Second Address - This register represents the most significant 32-bits of the address for a DAC transaction. Note: When the failing transaction used a single address cycle (SAC), then the contents of this register must be 0.



2.13.60 ECC Attribute Register - ECCAR

When the ECC Error Phase register (bits 6:4 of the ECCCSR) is non-zero (indicating that an error has been captured), the ECCAR register indicates the contents of the **P_AD[31:0]** bus (for 64- and 32-bit buses) for the attribute phase of the transaction that included the error. When the ECC Error Phase register is zero, the contents of this register are undefined.

Note: Registers that store information from the failing transaction always store information directly from the bus (uncorrected), even when correction of the error is possible.

Note: The "ECC Control and Status Register - ECCCSR", "ECC First Address Register - ECCFAR", "ECC Second Address Register - ECCSAR", and "ECC Attribute Register - ECCAR" report the actual transaction that has the error. For example, when the Split Completion of an original Outbound Read request had an error, the information regarding the Split Completion is reported.

Table 83. ECC Attribute Register - ECCAR

		<																															

2.13.61 CompactPCI Hot-Swap Capability ID Register

The following register block provides support of CompactPCI* Hot-Swap functionality.

Table 84. HS_CAPID - Hot-Swap Cap ID

Register Offset +0E8H																		Attribute Legend: RV = Reserved RW = Read/Write RO = Read Only				RT = Read/Toggle RC = Read/Clear SW = SROM Write NA = Not Accessible			
Bit	Default	Description																							
07:00	06h	Identifier (ID): 06h, identifying this linked list structure as being the Compact PCI Hot-Swap register block.																							



2.13.62 Offset EDh: HS_NXTP - Next Item Pointer

By default, the CompactPCI capability is the last capabilities list for the 81341 and 81342, thus this register defaults to 00H.

However, this register may be written to 90H prior to host configuration to include the VPD capability located at off-set 90H.

Warning: Writing this register to any value other than 00H (default) or 90H is not supported and may produce unpredictable system behavior.

In order to insure that this register is written prior to host configuration, the 81341 and 81342 must be initialized at **P_RST#** assertion to Retry Type 0 configuration cycles (bit 2 of PCSR). Typically, the Intel XScale® processor would be enabled to boot immediately following **P_RST#** assertion in this case (bit 1 of PCSR), as well. Please see [Section 2.13.41, “PCI Configuration and Status Register - PCSR”](#) on page 183 for more details on the 81341 and 81342 initialization modes.

Table 85. HS_NXTP - Next Item Pointer

Register Offset +0E9H Attribute Legend: RT = Read/Toggle RV = Reserved RC = Read/Clear RW = Read/Write SW = SROM Write RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	E8H	Next Capabilities Pointer (PTR): Since the cPCI capabilities are the last in the linked list of extended capabilities in the 81341 and 81342, the register is set to 00H. However, this field may be written prior to host configuration with 90H to extend the list to include the VPD extended capabilities header.



2.13.63 HS_CNTRL - Hot-Swap Control/Status Register

The 81341 and 81342 meets the standard requirements to be considered “Hot-Swap Silicon” detailed in the *Compact PCI Hot-Swap Specification*, Revision 2.1. Refer to the *Compact PCI Hot-Swap Specification*, Revision 2.1 for more details on the insertion and extraction processes.

Table 86. HS_CNTRL - Hot-Swap Control/Status Register (Sheet 1 of 2)

Register Offset +0EAH Attribute Legend: RV = Reserved RW = Read/Write RO = Read Only RT = Read/Toggle RC = Read/Clear SW = SRAM Write NA = Not Accessible		
Bit	Default	Description
07	1b (This bit actually powers up to 0b, however 81341 and 81342 sets it to 1b prior to any host software access to the device)	INS: Freshly INSerted board. 81341 and 81342 sets this bit to a 1b following the de-assertion of P_RST# provided that L_STAT is sampled low indicating that the ejector handle closed. The INS bit is cleared when software writes a 1b to it. Writing 0b to this bit has no effect. 1 = 81341 and 81342 asserts P_ENUM#, (when not masked by bit(1) of this register), to indicate that the card is freshly inserted and is ready to be configured by system software. After system software has cleared this bit (by writing a 1b to it) 81341 and 81342 de-asserts P_ENUM# (when currently asserted), and is then armed for a possible future extraction event (EXT bit assertion is enabled).
06	0b	EXT: Pending EXtraction of board. 81341 and 81342 sets this bit to a 1b when: (LOO = 0b or DHA = 0b), and - L_STAT is sampled high while P_RST# is deasserted indicating that the ejector handle is unlocked, and - The board is currently in the INSERTED state (i.e., the INS bit = 0b). The EXT bit is cleared when software writes a 1b. Writing a 0b has no effect. When 1b: 81341 and 81342 asserts P_ENUM#, (when not masked by bit(1) of this register), to indicate that the card is about to be removed.
05:04	01b	PI: Programming Interface This field is hard-wired to 01b indicating that 81341 and 81342 supports Device Hiding and PIE bit functionality.
03	0b	LOO: LED On/Off (LOO) Control. Allows software control of the LED. 0 = 81341 and 81342 drives LED_OUT low turning the external LED off. 1 = 81341 and 81342 drives LED_OUT high illuminating the external LED. Note: Additional external LED control logic must be ORed with 81341 and 81342 LED_OUT signal to ensure that the blue LED is illuminated while P_RST# is asserted or when the board is in the H0, H1, or H1F cPCI Hot-Swap defined hardware states.



Table 86. HS_CNTRL - Hot-Swap Control/Status Register (Sheet 2 of 2)

Register Offset +0E4H 7 4 0 IOP Attributes PCI Attributes Attribute Legend: RT = Read/Toggle RV = Reserved RW = Read/Write RO = Read Only RC = Read/Clear SW = SROM Write NA = Not Accessible		
Bit	Default	Description
02	0b	PIE: Pending Insertion/Extraction This status bit is set and cleared by the 81341 and 81342 Hot-Swap state Machine. When 1b this bit indicates that either an insertion or an extraction is in progress (either INS or EXT has a value of 1b or INS is armed).
01	0b	EIM: ENUM# Interrupt Mask. When 0b: 81341 and 81342 asserts P_ENUM# when an insertion or removal event occurs as indicated by the setting of the INS or EXT bits of this register. When 1b: 81341 and 81342 does not assert P_ENUM# under any circumstances.
00	0b	DHA: Device Hiding Armed: When 1b: When HS_SM# = 0b, and LSTAT = 1b (Switch open) and the LOO bit = 1b, 81341 and 81342 completes any bus cycles presently in process and then cease to initiate or respond to either primary or secondary bus cycles. When LSTAT subsequently goes low, 81341 and 81342 responds normally to bus cycles. When 0: 81341 and 81342 operates normally.



2.13.65 Inbound ATU Upper Base Address Register 3 - IAUBAR3

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR3 is set to indicate 32 bit addressability, the IAUBAR3 register attributes are read-only. By default the IAUBAR3 register has read-only attributes. Prior to changing the Type Indicator in the IABAR3 to support 32-bit addressability, the IAUBAR3 must be written with zero unless it already contains zero. Zero is the default value for IAUBAR3.

Table 88. Inbound ATU Upper Base Address Register 3 - IAUBAR3

Register Offset +204H		IOP Attributes PCI Attributes																															
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set																RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible															
Bit	Default	Description																															
31:0	00000H	Translation Upper Base Address 3 - Together with the Translation Base Address 3 these bits define the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes.																															



2.13.66 Inbound ATU Limit Register 3 - IALR3

Inbound address translation for memory window 3 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The inbound translation base address for inbound window 3 is specified in [Section 2.13.17](#). When determining block size requirements — as described in [Section 2.13.23](#) — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in [Section 2.2.1.1](#).

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR3 have a direct effect on the IABAR3 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR3 makes the corresponding bit within the IABAR3 a read only bit which always returns 0. A value of 1 in a bit within the IALR3 makes the corresponding bit within the IABAR3 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR3, all writes to the IABAR3 has no effect since a value of all zeros within the IALR3 makes the IABAR3 a read only register.

Table 89. Inbound ATU Limit Register 3 - IALR3

Register Offset +208H		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:12	00000H	Inbound Translation Limit 3 - This value determines the memory block size required for the ATUs memory window 3.																																			
11:00	000H	Reserved																																			



The Inbound ATU Translate Value Register 3 (IATVR3) in conjunction with the “[Inbound ATU Upper Translate Value Register 3 - IAUTVR3](#)” on [page 212](#) contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Register Offset +20CH Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:12	00000H	Inbound ATU Translation Value 3 - Bits 31 to 12 of internal bus address used to convert the PCI address to internal bus addresses. This value must be naturally aligned with the IABAR3 register's programmed value (see Section 2.13.23, "Determining Block Sizes for Base Address Registers" on page 169).
11:0:1	000H	Reserved
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read/write transactions through BAR3. When clear, no swap is performed. Refer to Section 2.3, "Big Endian Byte Swapping" on page 83 for more details.

The Inbound ATU Upper Translate Value Register 3 (IAUTVR3) in conjunction with the "Inbound ATU Translate Value Register 3 - IATVR3" on page 212 contain bits 35 to 12 of the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Register Offset
+210H

Attribute Legend:

- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RW = Read/Write
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



2.13.69 Outbound I/O Base Address Register - OIOBAR

The OIOBAR register locates the 64 KB I/O cycle address window in the 81341 and 81342's 64 Gbyte internal address space. When A[35:16] of the internal bus address matches the value in OIOBAR, the ATU claims the transaction and forward it over to the PCI interface as an I/O cycle.

Note: In translating the internal bus address A[35:0] for the PCI bus I/O cycle, A[15:0] is forwarded over to the PCI bus unmodified while A[31:16] is set to 0000H. (see "I/O Transactions" on page 76).

Table 92. Outbound I/O Base Address Register - OIOBAR

31 28 24 20 16 12 8 4 0 IOP Attributes PCI Attributes																													
		Register Offset +300H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																											
Bit	Default	Description																											
31:12	0 FFBH	Outbound I/O Base Address - This value represents bits 35 to 16 of the internal bus address used to claim an outbound I/O cycle request for the ATU.																											
11:3	000H	Reserved																											
2:0	0H	Outbound I/O Function Number Mapping - The Function number in this field is used as part of the Requestor ID for outbound I/O transactions																											



2.13.70 Outbound I/O Window Translate Value Register - OIOWTVR

The Outbound I/O Window Translate Value Register (OIOWTVR) contains the PCI I/O address used to convert the internal bus access to a PCI address. This address is driven on the PCI bus as a result of the outbound ATU address translation. See [Section 2.2.2.1, “Outbound Address Translation - Internal Bus Transactions”](#) on page 73 for details on outbound address translation.

The I/O window is from 81341 and 81342 internal bus is set via the “Outbound I/O Base Address Register - OIOBAR” with the a fixed length of 64 Kbytes.

Table 93. Outbound I/O Window Translate Value Register - OIOWTVR



2.13.71 Outbound Upper Memory Window Base Address Register 0 - OUMBAR0

The OUMBAR0 register locates Outbound Memory Window 0 in a 4 Gbyte Memory section in the 81341 and 81342 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR0[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the “Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0” on page 216.

Table 94. Outbound Upper Memory Window Base Address Register 0 - OUMBAR0

IOP Attributes PCI Attributes																													
		Internal Bus Address +308H																											
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																											
Bit	Default	Description																											
31	1 ₂	Outbound Window 0 Enable - When set, the ATU Outbound Memory Window 0 claims internal bus transactions when A[35:32] matches OUMBAR0[3:0].																											
30:28	000 ₂	Outbound Window 0 Function Number Mapping - Errors with Outbound Memory Window 0 transactions (PCI or Internal Bus) is associated with the function number contained in this field.																											
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 2.3, “Big Endian Byte Swapping” on page 83 for more details.																											
26:04	000 0000H	Reserved																											
3:0	1H	Outbound Upper Memory Window Base Address 0 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUE has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.																											



2.13.72 Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0

The Outbound Upper 32-bit Memory Window Translate Value Register 0 (OUMWTVR0) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

Table 95. Outbound Upper 32-bit Memory Window Translate Value Register 0-OUMWTVR0

		IOP Attributes PCI Attributes Register Offset +30CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																															
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).																															



2.13.73 Outbound Upper Memory Window Base Address Register 1 - OUMBAR1

The OUMBAR1 register locates Outbound Memory Window 1 in a 4 Gbyte Memory section in the 81341 and 81342 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR1[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the "Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1" on page 218.

Table 96. Outbound Upper Memory Window Base Address Register 1 - OUMBAR1

Register Offset +310H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31	1 ₂	Outbound Window 1 Enable - When set, the ATU Outbound Memory Window 1 claims internal bus transactions when A[35:32] matches OUMBAR1[3:0].
30:28	000 ₂	Outbound Window 1 Function Number Mapping - Errors with Outbound Memory Window 1 transactions (PCI or Internal Bus) is associated with the function number contained in this field.
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 2.3, "Big Endian Byte Swapping" on page 83 for more details.
26:04	000 0000H	Reserved
3:0	2H	Outbound Upper Memory Window Base Address 1 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUE has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.



The Outbound Upper 32-bit Memory Window Translate Value Register 1 (OUMWTVR1) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

Register Offset + 314H

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



2.13.75 Outbound Upper Memory Window Base Address Register 2 - OUMBAR2

The OUMBAR2 register locates Outbound Memory Window 2 in a 4 Gbyte Memory section in the 81341 and 81342 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR2[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the “Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2” on page 220.

Table 98. Outbound Upper Memory Window Base Address Register 2- OUMBAR2

IOP Attributes PCI Attributes																																	
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set																RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible															
Internal Bus Address +318H																																	
Bit	Default	Description																															
31	0 ₂	Outbound Window 2 Enable - When set, the ATU Outbound Memory Window 2 claims internal bus transactions when A[35:32] matches OUMBAR2[3:0].																															
30:28	000 ₂	Outbound Window 2 Function Number Mapping - Errors with Outbound Memory Window 2 transactions (PCI or Internal Bus) is associated with the function number contained in this field.																															
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 2.3, “Big Endian Byte Swapping” on page 83 for more details.																															
26:04	000 0000H	Reserved																															
3:0	3H	Outbound Upper Memory Window Base Address 2 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUE has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.																															



The Outbound Upper 32-bit Memory Window Translate Value Register 2 (OUMWTVR2) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

Internal Bus Address
+31CH

Attribute Legend:
 RW = Read/Write
 RC = Read Clear
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



2.13.77 Outbound Upper Memory Window Base Address Register 3 - OUMBAR3

The OUMBAR3 register locates Outbound Memory Window 3 in a 4 Gbyte Memory section in the 81341 and 81342 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR3[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the “Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3” on page 222.

Table 100. Outbound Upper Memory Window Base Address Register 3 - OUMBAR3

312824201612840 IOP Attributes PCI Attributes																																			
		Register Offset +320H																Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																	
Bit	Default	Description																																	
31	0 ₂	Outbound Window 3 Enable - When set, the ATU Outbound Memory Window 3 claims internal bus transactions when A[35:32] matches OUMBAR3[3:0].																																	
30:28	000 ₂	Outbound Window 3 Function Number Mapping - Errors with Outbound Memory Window 3 transactions (PCI or Internal Bus) is associated with the function number contained in this field.																																	
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 2.3, “Big Endian Byte Swapping” on page 83 for more details.																																	
26:04	000 0000H	Reserved																																	
3:0	4H	Outbound Upper Memory Window Base Address 3 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUE has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.																																	



The Outbound Upper 32-bit Memory Window Translate Value Register 3 (OUMWTVR3) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

Register Offset + 324H

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



2.13.79 Outbound Configuration Cycle Address Register - OCCAR

The Outbound Configuration Cycle Address Register is used to hold the 32-bit PCI configuration cycle address. The Intel XScale® processor writes the PCI configuration cycles address, which enables outbound configuration read or write. The Intel XScale® processor then performs a read or write to the Outbound Configuration Cycle Data Register to initiate the configuration cycle on the PCI bus.

Note: Bits 15:11 of the configuration cycle address for Type 0 configuration cycles are defined differently for Conventional versus PCI-X modes. When 81341 and 81342 software programs the OCCAR to initiate a Type 0 configuration cycle, the OCCAR should always be loaded based on the PCI-X definition for the Type 0 configuration cycle address. When operating in Conventional mode, the 81341 and 81342 clears bits 15:11 of the OCCAR prior to initiating an outbound Type 0 configuration cycle. See the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0 for details on the two formats.

Table 102. Outbound Configuration Cycle Address Register - OCCAR

Register Offset +330H																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																	
Bit	Default	Description																															
31:00	0000 0000H	Configuration Cycle Address - These bits define the 32-bit PCI address used during an outbound configuration read or write cycle.																															



The Outbound Configuration Cycle Data Register is used to initiate a configuration read or write on the PCI bus. The register is logical rather than physical meaning that it is an address not a register. The Intel XScale® processor reads or writes the data registers memory-mapped address to initiate the configuration cycle on the PCI bus with the address found in the OCCAR. For a configuration write, the data is latched from the internal bus and forwarded directly to the OWQ. For a read, the data is returned directly from the ORQ to the Intel XScale® processor and is never actually entered into the data register (which does not physically exist).

Table 103. Outbound Configuration Cycle Data Register - OCCDR

2.13.81 Outbound Configuration Cycle Function Number - OCCFN

For 81341 and 81342 the function number should be 0 for endpoint usage and match the ATUX function number (readable in "PCI-X Status Register - PCIXSR") for Root Complex modes.

[illegible]



2.13.82 PCI Interface Error Control and Status Register - PIECSR

This register indicates whether or not the ATU has detected and logged a PCI interface error. The register is also used to enable the logging of additional errors. For more details, see [Section 2.7, “ATU Error Conditions” on page 99](#).

Note: The “PCI Interface Error Control and Status Register - PIECSR”, “PCI Interface Error Address Register - PCIEAR”, and “PCI Interface Error Upper Address Register - PCIEUAR” report the original transaction when an error is detected on the current transaction. For example, when the Split Completion of an original Outbound Read request had an error, the information regarding the Outbound Read is reported.

Table 105. PCI Interface Error Control and Status Register - PIECSR

Register Offset +380H

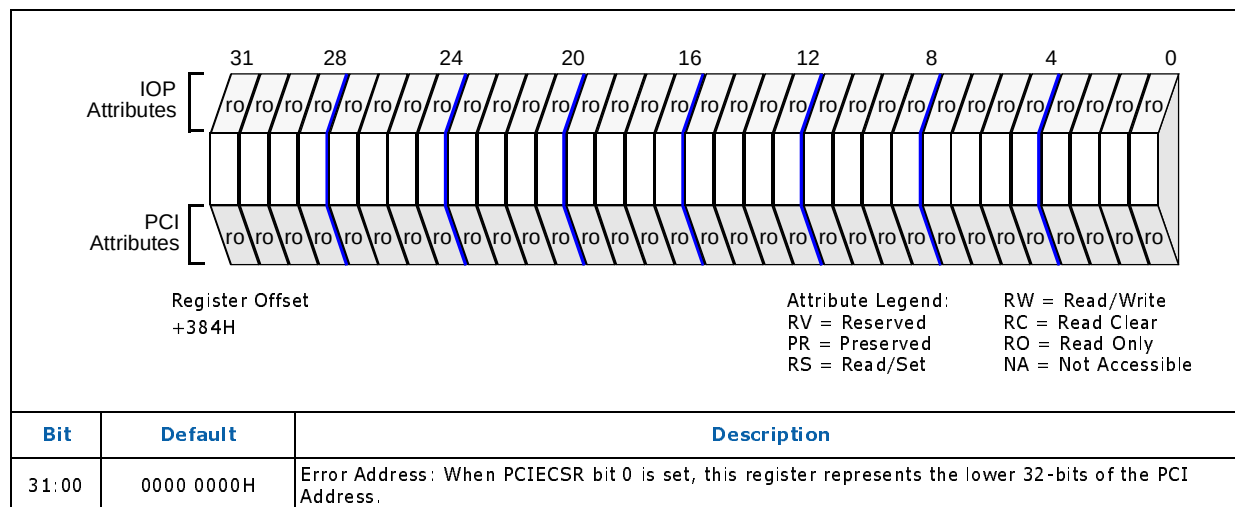
Attribute Legend:
 RV = Reserved
 RO = Read Only
 RC = Read Clear
 NA = Not Accessible
 RW = Read/Write
 RS = Read/Set

Bit	Default	Description
31:17	0000 0000H	Reserved
16:13	0H	Error Command - This field indicates the PCI command when PCIECSR bit 0 is set: This field represents the command type used during the PCI transaction other than the DAC command (when Dual Address Cycle). The 4-bit code in this field can be referenced to the actual command type via Table 2-8, "PCI-X Command Encoding" of the <i>PCI-X Protocol Addendum to the PCI Local Bus Specification</i> , Revision 2.0.
12:09	0000 ₂	Error Type - This field indicates the type of PCI interface error when PCIECSR bit 0 is set: 0000 Address Parity Error 0001 Data Parity Error 0010 Master Abort 0011 Target Abort 0100 Received Split Completion Error Message 0101 Unexpected Split Completion 0110 Split Completion Discarded 0111 Internal Bus Data Parity Error
08:05	0000 ₂	Initiator ID - When PCIECSR bit 0 is set, this Indicates the initiator associated with the detected PCI interface error. "0000" is used to denote inbound requests where the IOP was the completer of the transaction. See Internal Bus Requester IDs in the System Controller chapter for remaining details of Initiator ID.
04:02	000 ₂	PCI Function Number - When PCIECSR bit 0 is set, this field indicates the device function number that this error is associated with. Note: This field is undefined when the Error Type (bits 12:09) indicates an Address Parity Error.
01	0 ₂	Multiple PCI Interface Errors Detected - This bit is set when a PCI Interface Error has already been logged (bit 0 of the PCIECSR is set), but an additional error is detected. Note that a second error is logged for a subsequent transaction.
00	0 ₂	PCI Interface Error Detected - This bit is set when a PCI Interface Error is detected. When this register is cleared, the 81341 and 81342 is enabled to latch information about a PCI Interface error.



When PCIECSR bit 0 is set, this register represents the lower 32-bits of the address for the error detected on the PCI Bus. Note that for a DAC cycle the address may be 64-bit. This register is used in conjunction with [Section 2.13.84, “PCI Interface Error Upper Address Register - PCIEUAR” on page 227](#) in order to interpret the entire 64-bit PCI address for the error. One error can be detected and logged. The software knows which PCI address had the error by reading this register and decoding the contents of the PCIECSR. For error details, see [Section 2.7, “ATU Error Conditions” on page 99](#)).

Table 106. PCI Interface Error Address Register - PCIEAR





2.13.84 PCI Interface Error Upper Address Register - PCIEUAR

When PCIECSR bit 0 is set and the PCI error detected included a DAC cycle, this register represents the upper 32-bit address of where the error was detected on the PCI bus. This register is used in conjunction with the [Section 2.13.83, "PCI Interface Error Address Register - PCIEAR"](#) on page 226. One error can be detected and logged. The software knows which PCI address had the error by reading this register and decoding contents of the PCIECSR. For error details, see [Section 2.7, "ATU Error Conditions"](#) on page 99).

Note: The "PCI Interface Error Control and Status Register - PIECSR", "PCI Interface Error Address Register - PCIEAR", and "PCI Interface Error Upper Address Register - PCIEUAR" report the original transaction when an error is detected on the current transaction. For example, when the Split Completion of an original Outbound Read request had an error, the information regarding the Outbound Read is reported.

Table 107. PCI Interface Error Upper Address Register - PCIEUAR

312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Register Offset +388H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	0000 000H	Upper 32-bit Address - When bit 0 of the PCIECSR is set, this register represents the upper 32 bits of the PCI Address.



2.13.85 PCI Interface Error Context Address Register — PCIECAR

When PCIECSR bit 0 is set, this register contains the DMA Channel Number and bits 30 through 5 of the address of the ADMA descriptor associated with the error detected on the PCI Bus. One error can be detected and logged. The software knows which ADMA descriptor context had the error by reading this register and decoding the contents of the PCIECSR. For error details, see [Section 2.7, "ATU Error Conditions" on page 99](#)).

Table 108. PCI Interface Error Context Address Register - PCIECAR

		31	28	24	20	16	12	8	4	0														
IOP Attributes	[	rv	ro	ro	ro	ro	ro	ro	ro	ro	ro	]												
		ro	ro	ro	ro	ro	ro	ro	ro	ro	ro													
PCI Attributes	[	rv	ro	ro	ro	ro	ro	ro	ro	ro	ro	]												
		ro	ro	ro	ro	ro	ro	ro	ro	ro	ro													
		Register Offset +38CH																						
															Attribute Legend:									
															RW = Read/Write									
															RV = Reserved									
															RC = Read Clear									
															PR = Preserved									
															RO = Read Only									
															RS = Read/Set									
															NA = Not Accessible									
Bit	Default	Description																						
31	0	Reserved																						
30:29	00	DMA Channel Number																						
28:0	0H	ADMA Descriptor Address Bits 30:5																						



2.13.86 Internal Arbiter Control Register - IACR

The Internal Arbiter Control Register is used to control which priority ring different PCI bus requesters (including the ATU) use. In addition, the method by which the arbiter parks on masters is configurable in this register.

Table 109. Internal Arbiter Control Register - IACR

Register Offset +394H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:9	0 ₂	Reserved.
08	0 ₂	Bus Parking Control: Controls the bus parking behavior of the internal arbiter: 0 = Bus is parked on the last PCI agent using the bus. 1 = Bus is always parked on 81341 and 81342.
07	0 ₂	Reserved
06	0 ₂	ATU Ring Allocation: Priority ring allocation for 81341 and 81342 ATU requests: 0 = The ATU is in the low priority ring of the internal arbiter. 1 = The ATU is in the high priority ring of the internal arbiter.
05:04	00 ₂	Reserved.
03:00	0000 ₂	PCI Master Priority Ring Allocation: Bit 0 corresponds to REQ#[0], 1 corresponds to REQ#[1] and so on: 0 = The corresponding master is in the low priority ring of the internal arbiter. 1 = The corresponding master is in the high priority ring of the internal arbiter.



2.13.87 Multi-Transaction Timer - MTT

This register controls the amount of time that the 81341 and 81342 arbiter allows a PCI initiator to perform multiple back-to-back transactions on the PCI bus. The number of clocks programmed in the MTT represents the insured time slice (measured in PCI clocks) allotted to the current agent, after which the arbiter grants another agent that is requesting the bus.

Table 110. Multi-Transaction Timer - MTT

Register Offset +398H 7 4 0 IOP Attributes rw rw rw rw rw rv rv rv PCI Attributes ro ro ro ro ro rv rv rv Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:03	00H	Timer Count Value (MTC): This field specifies the amount of time that grant remains asserted to a master continuously asserting its request for multiple transfers. This field specifies the count in an 8-clock (PCI clock) granularity.
02:00	0H	Reserved



2.13.88 PCIX RCOMP Control Register — PRCR

Warning: In Central Resource mode this register can only be accessed when PCSR bit 21 is cleared. Refer to Table 64, “PCI Configuration and Status Register - PCSR” on page 183.

Table 111. PCIX RCOMP Control Register - PRCR

Intel XScale® processor Local Bus offset +2100H Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:12	00000H	Reserved
11	0 ₂	When this bit is set, the slew rate manual override values are used. When this bit is cleared, the slew rate is derived from the drive strength (normal operation).
10:09	00 ₂	Reference select for PCIX CAP pad 2'b00 Nominal Reference Voltage 2'b01 -5% Reference 2'b10 +5% Reference
08:07	01 ₂	Drive strength select for ODT RCOMP pad 2'b00 103.0 ohms 2'b01 110.0 ohms 2'b10 120.0 ohms
06:04	010 ₂	Drive strength select for RCOMP pad dedicated 3.3V supply voltage 3'b000 18.0 ohms 3'b001 20.1 ohms 3'b010 22.0 ohms 3'b011 24.1 ohms 3'b100 26.1 ohms
03:01	001 ₂	Drive strength select for RCOMP pad 1.5V switch supply voltage 3'b000 18.0 ohms 3'b001 20.1 ohms 3'b010 22.0 ohms 3'b011 24.1 ohms 3'b100 26.1 ohms
00	1 ₂	RCOMP pads enable.



2.13.89 PCIX Pad ODT Drive Strength Manual Override Values Registers — PPODSMOVR

Warning: In Central Resource mode this register can only be accessed when PCSR bit 21 is cleared. Refer to Table 64, “PCI Configuration and Status Register - PCSR” on page 183.

Table 112. PCIX Pad ODT Drive Strength Manual Override Values Registers - PPODSMOVR

		31	28	24	20	16	12	8	4	0			
IOP Attributes	[	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	
		rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	
PCI Attributes	[	na	na	na	na	na	na	na	na	na	na	na	
		na	na	na	na	na	na	na	na	na	na	na	
Intel XScale® processor Local Bus offset +2104H												Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description											
31:14	00000H	Reserved											
13:08	100000 ₂	N-ODT drive strength manual override values for PCIX pad.											
07:06	00 ₂	Reserved.											
05:00	100000 ₂	P-ODT drive strength manual override values for PCIX pad.											



2.13.91 PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V Dedicated Supply Voltage) — PPDSMOVR3.3

Warning: In Central Resource mode this register can only be accessed when PCSR bit 21 is cleared. Refer to Table 64, “PCI Configuration and Status Register - PCSR” on page 183.

Table 114. PCIX PAD DRIVE STRENGTH Manual Override Values Register (3.3 V Dedicated Supply Voltage) — PPDSMOVR3.3

		31	28	24	20	16	12	8	4	0				
IOP Attributes	[	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv		
		rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv		
PCI Attributes	[	na	na	na	na	na	na	na	na	na	na	na		
		na	na	na	na	na	na	na	na	na	na	na		
Intel XScale® processor Local Bus offset +210CH												Attribute Legend:	RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set	RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
31:28	0H	Reserved
27:24	1000 ₂	N-slew rate manual override values for PCIX pad.
23:20	0000 ₂	Reserved.
19:16	1000 ₂	P-slew rate manual override values for PCIX pad.
15:14	00 ₂	Reserved.
13:08	100000 ₂	N-drive strength manual override values for PCIX pad.
07:06	00 ₂	Reserved.
05:00	100000 ₂	P-drive strength manual override values for PCIX pad.



3.0 Address Translation Unit (PCI Express)

This chapter describes the operation modes, setup, and implementation of the module which interfaces between the PCI Express Link and the Intel® 81341 and 81342 I/O Processors (81341 and 81342) internal bus.

3.1 Overview

As indicated in [Figure 23](#), the Address Translation Unit (ATU) — the interface between the PCI Express Link and the on-chip internal bus — consists of the Address Translation Unit (ATU) and the Expansion ROM Unit.

The ATU supports both inbound and outbound address translation. The ATU provides access between the PCI Express Link and the 81341 and 81342 internal bus.

Transactions initiated on the PCI Express Link and targeted at the 81341 and 81342 internal bus are referred to as *inbound transactions* (PCI Express to internal bus). Transactions initiated on the 81341 and 81342 internal bus and targeted at the PCI Express Link are referred to as *outbound transactions* (internal bus to PCI Express). The ATU accepts multiple inbound or outbound transactions and processes them simultaneously.

During inbound transactions, the ATU converts PCI addresses (initiated by a PCI Express Requester) to internal bus addresses and initiates the data transfer on the 81341 and 81342 internal bus. During outbound transactions, the ATU converts internal bus addresses to PCI addresses and initiates the data transfer on the PCI Express Link.

The Expansion ROM provides the PCI mechanism for downloading device/board driver code during system boot sequence. It consists of a separate inbound address range which accesses a Flash EPROM device connected through the 81341 and 81342 memory controller. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details of Expansion ROM usage.

The Address Translation Unit and the Expansion ROM Translation Unit represent a single function of the multi-function 81341 and 81342 device.

The ATU supports the following PCI Express Lane widths and frequencies delivering up to 4096 Mbytes/sec of bandwidth:

- Lane Widths: x8, x4, x2, x1
- Link Frequency: 2.5Gbits/s

All PCI Express transactions are protected by link layer CRC.

On the internal interface, the ATU implements the 81341 and 81342 internal bus protocol which provides for a maximum of 4800 Mbytes/sec of bandwidth.

Address and data are protected by byte-wise parity on the internal bus.



The ATU includes four capability headers that implement Power Management capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1, MSI, MSI-X, and Vital Private Data (VPD) capabilities as defined by *PCI Local Bus Specification*, Revision 2.3, and PCI Express capability as defined by *PCI Express Base Specification*, Revision 1.0a.

Additionally, the ATU includes three PCI Express Extended capability headers that implement Advanced Error Handling, Device Serial Number, and Power Budgeting as defined in the *PCI Express Base Specification*, Revision 1.0a

The functionality of the ATU is described in the following sections. The ATU has a memory-mapped register interface that is visible from either the PCI interface, the internal bus interface, or both.

Figure 23. ATU Block Diagram

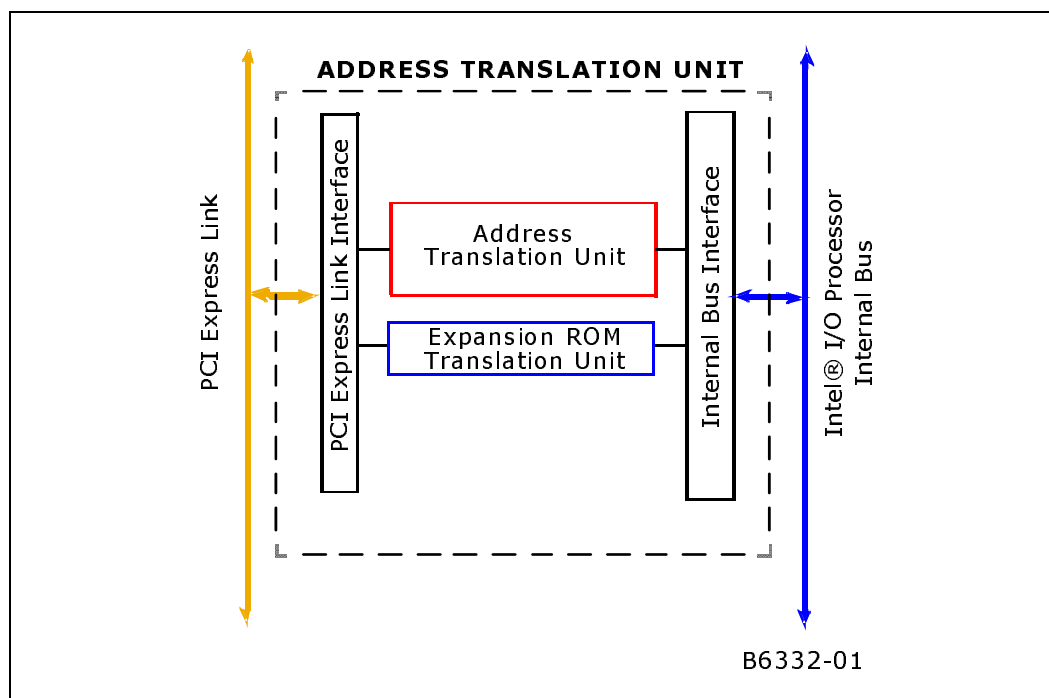
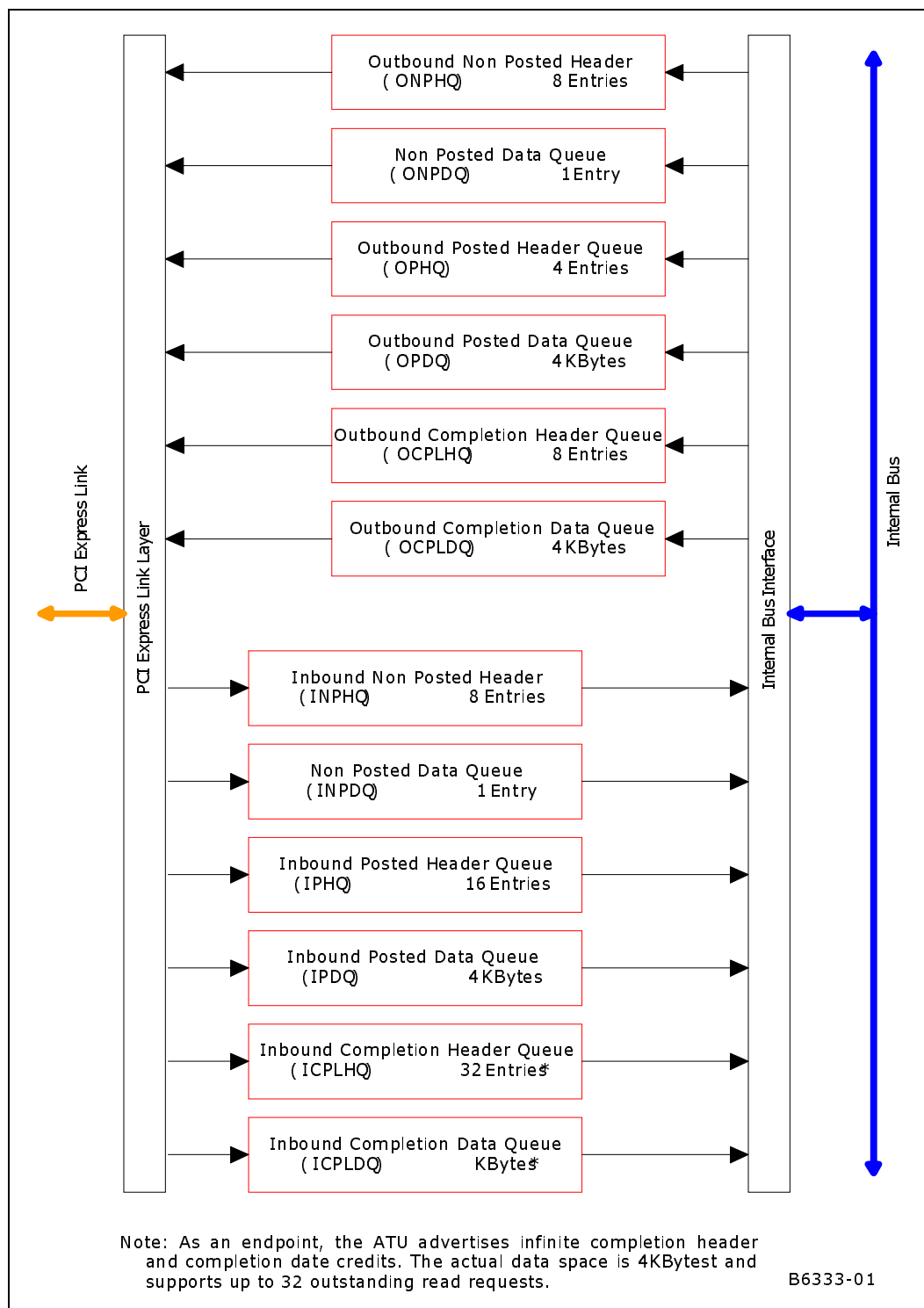




Figure 24. ATU Queue Architecture Block Diagram





3.2 PCI Express* Link Characteristics

The PCI Express** port supports x8, x4, x2, and x1 operation. Lane reversal and polarity inversion automatically occur in an attempt to successfully train the link.

The PCI Express* port is configured to ease adapter card implementations. The lane number and lane polarity should enable straight routing between the component and the PCI Express* card edge connector. The lane reversal feature can be utilized to simplify applications where this component is connected to the upstream device in a planar fashion. For example integrating the device on a motherboard with direct connection to the I/O Hub.

Supported Lane Reversal Modes

x8	7:0, 0:7
x4	3:0, 0:3
x2	1:0, 0:1
x1	0, 7

Port bifurcation is not supported. This component supports a single x8 PCI Express port and can not be split into multiple x4 ports.

The PCI Express interface supports a maximum payload size of 512 Bytes and returns completions with minimum of 128 byte payloads.



3.3 ATU Address Translation

The ATU allows PCI Express requesters to initiate transactions to the 81341 and 81342 internal bus and allows the Intel XScale® processor to initiate transactions in the PCI Express domain.

The ATU implements an address windowing scheme to determine which addresses to claim and translate to the destination bus.

- The address windowing mechanism for inbound translation is described in [Section 3.3.1.1, “Inbound Address Translation” on page 242](#).
- The address windowing mechanism for outbound translation is described in [Section 3.3.2, “Outbound Transactions” on page 249](#) and [Section 3.3.3, “Outbound Write Transaction” on page 255](#).

The ATU has the ability to accept up to eight inbound PCI Express Non Posted Read (memory read, configuration read, and I/O read) transactions, one inbound Non Posted Write (configuration write and I/O write) transaction, and 16 inbound PCI Express Posted (memory write and message) transactions simultaneously.

As a PCI Express end point, the ATU must advertise infinite credits for Completion Headers and Completion Data. As a requester the ATU never requests more read data than it has room for in its Inbound Completion queues (ICPLHQ and ICPLDQ).

Also, the ATU has the ability to accept up to eight outbound Non Posted (internal bus read, configuration, and I/O) transactions and four outbound Posted (internal bus write and message) transactions simultaneously.

Of the 8 outbound Non Posted transactions, 4 is actively requesting data while the other 4 remains pending until one of the earlier active transactions is completed. Each active outbound read request may be fragmented into sub-requests based on the MAX_READ_REQUEST_SIZE parameter programmed in the “[PCI Express Device Control Register - PE_DCTL](#)” on page 348. The fragmentation may result in as many as 8 sub-requests per active read transaction. The ATU tracks a maximum of 32 outstanding non posted transactions at once. Completions for these sub-requests can return out of order on the PCI Express interface but they are returned in order on the internal bus.

Outbound memory writes and completions may be fragmented into smaller transactions based on the setting of the MAX_PAYLOAD_SIZE in the “[PCI Express Device Control Register - PE_DCTL](#)” on page 348.

Outbound completions obeys the minimum fragmentation limit of 128Bytes.

Refer to [Figure 24](#) and [Section 3.8](#) for details of the ATU queue architecture.

As a master on the internal bus, the ATU never requests more read data than it has room for in the Outbound Completion queues (OCPLHQ and OCPLDQ).

Inbound memory writes are fragmented on 1KB address aligned boundaries before issuing on the internal bus. Since the maximum payload size supported by the PCI Express interface is 512 Bytes each transactions are fragmented into a maximum of two internal bus transactions. Additionally, write combining does not occur.

Inbound completions are attempted on the internal bus with the same payload size as was received from PCI Express. In most instances the PCI Express completion size is 64B or 128B. The PCI express completions may be combined into larger completion transactions on the internal bus.

Inbound memory read requests are fragmented into 1KB aligned sub-requests. Completions for these sub-requests can be received out of order on the internal bus and is returned in order on the PCI Express interface.



The ATU unit allows for recognition and generation of multiple PCI Express Transaction Layer Packets (TLP) types. [Table 116](#) shows the commands supported for both inbound and outbound ATU transactions. The type of operation seen by the ATU on inbound transactions are determined by the PCI Express requester who initiates the transaction. Claiming an inbound transaction depends on the address range programmed within the inbound translation window. The type of transaction used by the ATU on outbound transactions generated by the core processor is determined by the internal bus address and the fixed outbound windowing scheme.

ATU supports all four address spaces defined within the PCI Express architecture as both a requester and completer. These address spaces and corresponding transaction types are detailed in [Table 115](#).

Table 115. Supported Address Spaces and Transaction Types

Address Space	Transaction Type	Basic Usage
Memory ^a	Read Write	Transfer data to/from a memory-mapped location
I/O	Read Write	Transfer data to/from an I/O-mapped location
Configuration	Read Write	Device configurations/setup
Message	Baseline Vendor-defined	From event signalling mechanism to general purpose messaging

a. ATU supports both 32-bit and 64-bit addressing for Memory space transactions.

ATU does not support Locked Requests as a Completer nor generate them as a Requestor.

**Table 116. ATU Command Support**

TLP Type	Fmt[1:0] ^a	Type[4:0]	Supported as Completer	Generated as Requester	Valid Internal Bus Command
MRd	00 01	0 0000	Yes	Yes	Read
MRdLk	00 01	0 0001	Unsupported Request	No	N/A
MWr	10 11	0 0000	Yes	Yes	Write
IORd	00	0 0010	Yes	Yes	Read
IOWr	10	0 0010	Yes	Yes	Write
CfgRd0	00	0 0100	Yes	Yes	Read
CfgWr0	10	0 0100	Yes	Yes	Write
CfgRd1	00	0 0101	Unsupported Request	Yes	N/A
CfgWr1	10	0 0101	Unsupported Request	Yes	N/A
Msg	01	1 0r ₂ r ₁ r ₀	Yes	Yes	Write ^b
MsgD	11	1 0r ₂ r ₁ r ₀	Yes	Yes	Write ^b
Cpl	00	0 1010	Yes	Yes	Completion
CplD	10	0 1010	Yes	Yes	Completion
CplLk	00	0 1011	Unexpected Completion ^c	No	N/A
CplDLk	10	0 1011	Unexpected Completion ^c	No	N/A
	All encodings not shown above are reserved		Malformed Packet	No	N/A

a. Requests with two Fmt[1:0] values shown can use either 32b (the first value) or 64b (the second value) Addressing Packet formats.

b. As a completer, the ATU stores the message header and payload directly in memory mapped registers. No internal bus traffic occurs.

c. ATU does not generate lock requests and any CplLk or CplDLk would be an unexpected completion.

Inbound and outbound ATU transactions are best described by the data flows used on the PCI Express Link and the 81341 and 81342 internal bus during read and write operations. The following sections describe read and write operations for inbound ATU transactions (PCI Express to internal bus) and outbound transactions (internal bus to PCI Express). All transactions are full split and the requests and completions are described separately.

3.3.1 Inbound Transactions

Inbound transactions are received on the PCI Express receive port and forward to the 81341 and 81342 internal bus. This transactions include all requests for which the ATU is the completer as well as completions for which the ATU was the initiator.

Inbound request transactions which target the ATU are translated and executed on the 81341 and 81342 internal bus. As a PCI Express completer, the ATU is capable of accepting all memory, I/O, and configuration request. Additionally, as a PCI Express end-point, the ATU would never request more memory read data than it can hold in its Inbound Completion Data Queue.

Inbound memory write (and message) transactions have their headers entered into the inbound posted header queue (IPHQ) and data entered into the inbound posted data queue (IPDQ). The IPHQ/IPDQ pair are capable of holding up to 16 posted operations up to the size of the data queue. Inbound configuration (or I/O) write transactions use the inbound non posted header queue (INPHQ) and inbound non posted data queue (INPDQ). The INPDQ has room for one configuration or I/O write at a time. Refer to [Section 3.8](#) for details of queue operation. Inbound read transaction (memory, configuration, and I/O) have their header entered into the inbound non posted header queue (INPHQ) and the data are returned to the PCI Express requester in the outbound completion data queue (OCPLDQ). The INPQ is capable of holding up to 8 non posted requests and any associated data.

Operation of the internal bus is defined in [Section 6.0, “System Controller \(SC\) and Internal Bus Bridge”](#). PCI Express has three principal mechanisms for Transaction Layer Packet (TLP) routing: address, ID, and implicit. The following sections describes how the ATU routes and translates each type.

3.3.1.1 Inbound Address Translation

PCI Express utilizes both 32-bit and 64-bit address schemes via the 3DW and 4DW headers. To prevent address aliasing, all devices must decode the entire address range. All discussions in this section refer to 64-bit addressing. When the 3DW header is used the upper 32-bits of address are assumed to be 0000_0000h.

The ATU allows external PCI Express requesters to directly access the internal bus via address routed TLPs. These PCI Express requesters can read or write 81341 and 81342 memory-mapped registers or 81341 and 81342 local memory space. The process of inbound address translation involves two steps:

1. Address Detection.
 - Verify the PCI address is within the address windows defined for the inbound ATU.
 - When the address is outside of the ATU address registers, the transaction is terminated as an unsupported request (UR).
2. Address Translation.
 - Translate the lower 32-bit PCI address to a 36-bit 81341 and 81342 internal bus address.

The ATU uses the following registers in inbound address window 0 translation:

- [Section 3.16.13, “Inbound ATU Base Address Register 0 - IABAR0” on page 308](#)
- [Section 3.16.28, “Inbound ATU Limit Register 0 - IALR0” on page 322](#)
- [Section 3.16.29, “Inbound ATU Translate Value Register 0 - IATVR0” on page 323](#)
- [Section 3.16.30, “Inbound ATU Upper Translate Value Register 0 - IAUTVR0” on page 323](#)



The ATU uses the following registers in inbound address window 1 translation:

- [Section 3.16.16, “Inbound ATU Base Address Register 1 - IABAR1” on page 312](#)
- [Section 3.16.31, “Inbound ATU Limit Register 1 - IALR1” on page 324](#)
- [Section 3.16.32, “Inbound ATU Translate Value Register 1 - IATVR1” on page 325](#)
- [Section 3.16.33, “Inbound ATU Upper Translate Value Register 1 - IAUTVR1” on page 325](#)

The ATU uses the following registers in inbound address window 2 translation:

- [Section 3.16.18, “Inbound ATU Base Address Register 2 - IABAR2” on page 314](#)
- [Section 3.16.34, “Inbound ATU Limit Register 2 - IALR2” on page 326](#)
- [Section 3.16.35, “Inbound ATU Translate Value Register 2 - IATVR2” on page 327](#)
- [Section 3.16.36, “Inbound ATU Upper Translate Value Register 2 - IAUTVR2” on page 328](#)

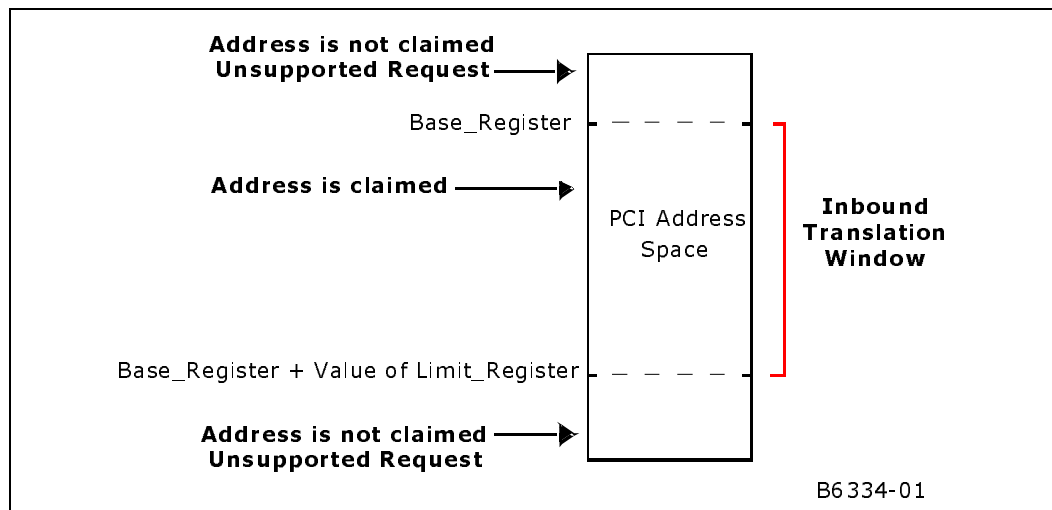
Inbound address detection is determined by comparing the 64-bit PCI address with the base address register and the limit register. In the case of 3DW headers, the upper 32-bits of the address is assumed to be 0000_0000h during address comparison. The algorithm for detection is:

Equation 8. Inbound Address Detection

When $\text{PCI\_Address}[31:0] \& \text{Limit\_Register}[31:0] == \text{Base\_Register}[31:0]$ and $\text{PCI\_Address}[63:32] == \text{Base\_Register}[63:32]$ the PCI Address is translated by the Inbound ATU. Otherwise treat as an Unsupported Request.

Figure 25 shows an example of inbound address detection.

Figure 25. Inbound Address Detection



The lower 32-bits of the incoming address is bitwise ANDed with the associated inbound limit register. When the result matches the base register, the inbound PCI address is detected as being within the inbound translation window and is claimed by the ATU. When the address is outside the translation window, the ATU terminates the transaction as an Unsupported Request (UR).

Note: By default, the first 8Kbytes of the ATU inbound address translation window 0 are reserved for the Messaging Unit. See [Section 3.5, “Messaging Unit” on page 261](#).

Once the transaction is claimed, the upper 32-bits of the address is discarded and the lower 32-bits of the address must be translated from a PCI address to a 36-bit internal bus address. The algorithm is:

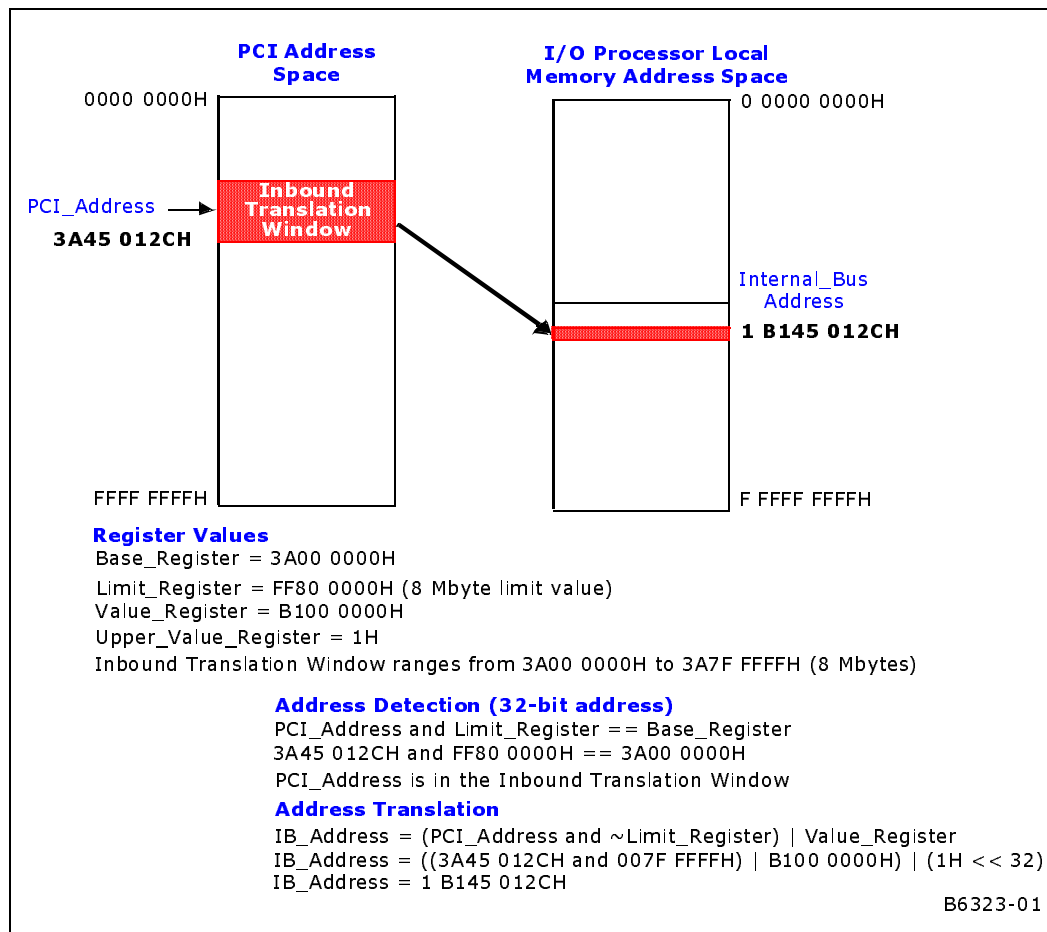
Equation 9. Inbound Translation

$$81341 \text{ and } 81342 \text{ Internal Bus Address} = ((\text{PCI\_Address}[31:0] \& \sim \text{Limit\_Register}[31:0]) \mid \text{ATU\_Translate\_Value\_Register}[31:0]) \mid (\text{ATU\_Upper\_Translate\_Value\_Register}[3:0] \ll 32).$$

The lower 32-bits of the incoming PCI address are first bitwise ANDed with the bitwise inverse of the limit register. This result is bitwise ORed with the ATU Translate Value, which is then ORed with the 4-bit ATU Upper Translate Value left shifted by 32; the result is the 36-bit internal bus address. This translation mechanism is used for all inbound memory read and write commands excluding inbound configuration read and writes. Inbound configuration cycle translation is described in [Section 3.3.1.5, "Inbound Configuration Cycle Translation \(ID Routed\)"](#) on page 247.

Figure 26 shows an inbound translation example for 32-bit addressing. This example would hold true for an inbound transaction from PCI Express Link.

Figure 26. Inbound Translation Example





3.3.1.2 Inbound Memory Write Transaction

An inbound write transaction is initiated by a PCI Express requester and is targeted at either 81341 and 81342 local memory or a 81341 and 81342 memory-mapped register.

Data flow for an inbound write transaction is summarized as:

- The ATU accepts the write transaction when the PCI address is within one of the inbound translation windows defined by the ATU Inbound Base Address Register, Inbound Upper Base Address Register, and Inbound Limit Register.
- When the IPHQ is full or the IPDQ overflows, a flow control error occurred and an ERR_FATAL is returned to the root complex.

Once the inbound write packet has passed all the TLP validation checks, the ATUs internal bus interface becomes aware of the inbound write. When there are additional write transactions ahead in the IPHQ, the current transaction remains posted until ordering and priority have been satisfied (Refer to [Section 3.8.3](#)) and the transaction is attempted on the internal bus by the ATU internal master interface.

Data flow for the inbound write transaction on the internal bus is summarized as:

- The ATU internal bus master requests the internal bus when IPHQ has at least one entry.
- When the internal bus is granted, the internal bus master interface initiates the write transaction by driving the translated address onto the internal bus. For details on inbound address translation, see [Section 3.3, "ATU Address Translation" on page 239](#).
- When an internal bus target does not claim write transaction, a master abort condition is signaled on the internal bus. The current transaction is flushed from the queue and an unsupported request (UR) message may be generated on the PCI Express interface.
- The ATU initiator interface attempts a 128-bit wide transfer on the internal bus. When the target that claims the request does not support 128-bit wide transfers, a 64-bit wide transfer is used. Transfers use internal bus byte enables to mask the bytes not written in each data phase. Write data is transferred from the IPDQ to the internal bus while data is available and the internal bus interface retains internal bus ownership. Refer to [Section 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#) for details of internal bus operation.
- The internal bus interface stops transferring data from the current transaction to the internal bus when one of the following conditions becomes true:
 - The internal bus initiator interface loses bus ownership.
 - The data from the current transaction has completed (satisfaction of payload length). An initiator termination is performed and the bus returns to idle.
 - A Master Abort is signaled on the internal bus. Data is flushed from the IPDQ.

3.3.1.3 Inbound Memory Read Transaction

An inbound read transaction is initiated by a PCI Express requester and is targeted at either local memory or a memory-mapped register space. The read transaction is propagated through the inbound non posted queue (INPQ) and read data is returned through outbound completion data and header queues (OCPLHQ, OCPLDQ).

In PCI Express, all read transactions are processed as split transactions. The ATUs PCI Express interface accepts the read transaction and forwards the read request through to the internal bus and returns the read data to the PCI Express Link. Data flow for an inbound read transaction is summarized in the following statements:

- The ATU accepts the read transaction when the PCI address is within one of the inbound translation windows defined by ATU Inbound Base Address Register, Inbound Upper Base Address Register, and Inbound Limit Register.
- When the transaction crosses a 1KB aligned boundary it is fragmented into smaller requests that do not cross the aligned boundary before it is issued on the internal bus. Since PCI Express transactions cannot cross a 4KB boundary, a single read request is broken into at most 4 1KB transactions.
- When sufficient space exists in OCPLDQ, request internal bus and issue request.
- Save the completion header information in the OCLUT.
- ATUE can handle a maximum of 4 outstanding internal bus requests at one time.
- All internal bus read requests result in split completions. The completion data is queued in the Outbound Completion Data Queue.
- A zero length read (memory read request of 1 DW with no bytes enabled) has no side-effects.
- Once a completion transaction has started, it continues until one of the following is true:
 - The length is satisfied.
 - An internal bus Master Abort or Target Abort was detected. The ATU generates a Completion TLP with a Completer Abort status to inform the requester about the abnormal condition. The INPHQ for this transaction is flushed. Refer to [Section 3.9.3](#).

The data flow for an inbound read transaction on the internal bus is summarized in the following statements:

- The ATU internal bus master interface requests the internal bus when a PCI address appears in an INPHQ and transaction ordering has been satisfied. The ATU takes advantage of the information provided by the Relaxed Ordering Attribute bit.
- Once the internal bus is granted, the internal bus master interface drives the translated address onto the bus. When a Retry is signaled, the request is repeated. When a master abort occurs, transaction is considered complete and an unsupported request Completion is loaded into OCPLHQ for return to the PCI Express requester (request is flushed once the completion has been posted to the OCPLHQ).
- Once the translated address is on the bus and the transaction has been claimed, the internal bus target starts returning data using a split response. Read data is continuously received by the OCPLDQ until one of the following is true:
 - The full byte count requested by the ATU read request is received. The internal bus completer's initiator interface performs an initiator completion in this case.
 - A partial byte count requested by the ATU read request is received. The completer's internal bus initiator interface performs an initiator completion in this case. Also, the completer reacquires the internal bus to deliver the remaining read data byte count to the ATU.



3.3.1.4 Inbound I/O Cycle Translation

Inbound address window 2 can be configured to accept I/O Read and I/O Write transactions by setting the Memory/IO space indicator bit to 1 in the [“Inbound ATU Base Address Register 2 - IABAR2”](#) on page 314. All I/O cycles are 32-bit transactions (DWORD).

For inbound I/O reads, the INPQ is used in the same manner as inbound memory read operations and the exception cases are identical. However, the internal bus cycle that results is always be a 32-bit transaction.

For inbound I/O writes, the ATU uses the INPHQ to hold both the header and the INPDQ to hold the data. An I/O write TLP with poisoned data results in the data being dropped and a Completion with Completions status of UR is returned to the requester.

When there are no errors during the request cycle, transaction ordering and priority are satisfied. Next, the internal bus master interface requests the internal bus and deliver the write data to the target as defined in [Section 3.3.1.2](#).

The status of the transaction on the internal bus is returned to the requestor on the PCI Express Link. When the Write Cycle Master Aborts on the Internal Bus, a Completion with status of Completer Abort is returned.

3.3.1.5 Inbound Configuration Cycle Translation (ID Routed)

The 81341 and 81342 ATU only accepts Type 0 configuration requests with a function number of zero when bit 7 of the ATUHTR (see [Section 3.16.11, “ATU Header Type Register - ATUHTR”](#) on page 306) is cleared or function numbers of zero and one when bit 7 of the ATUHTR is set.

The ATU is a native PCI Express device that supports both the standard PCI express capability structure and the PCI Express Extended capability structure. The ATU configuration space is selected by any PCI Express Type 0 configuration cycle targeting function 0. The bus number and device number is captured from all valid Type 0 configuration write TLPs that target an enabled function.

For inbound configuration reads, the INPQ is used in the same manner as inbound memory read operations.

For inbound configuration writes, the ATU uses the INPQ to hold both the header and the data. An configuration write TLP with poisoned data results in the data being dropped and an Completion with Completions status of UR is returned to the requester.

When there are no errors during the request cycle, transaction ordering and priority are satisfied. Next, the internal bus master interface requests the internal bus and deliver the write data to the target as defined in [Section 3.3.1.2](#).

Since Master Aborts and Target Aborts cannot occur during configuration cycles on the internal bus, a Completion TLP with Successful Completion (SC) status is generated for all configuration cycles.

When the Configuration Request Retry bit is set (PCSR[2]), the Configuration Request cycle is terminated with a Configuration Request Retry Status (CRS).



3.3.1.6 Inbound Vendor_Defined Message Transactions

Inbound messages are routed to the PCI Express message unit where they are decoded and processed.

Inbound Vendor_Defined Messages (IVM) are logged in the Inbound Message Header0-3 and Inbound Message Payload registers and an interrupt is conditionally sent to the Intel XScale® processor.

Only one message can be pending in the Inbound Vendor Message registers at one time. When bit 6 of the “[ATU Configuration Register - ATUCR](#)” on page 330 is set, then subsequent IVM are dropped. This is necessary to prevent deadlock when the Intel XScale® processor has outstanding read transactions.

When bit 6 is cleared, then when a second vendor specific message transaction reaches the head of the IPHQ it stalls until the message registers are freed by clearing the Message Received bit in the ATUISR. Since messages are posted transactions, they stall all other transactions until they make progress.

When the message received interrupt mask is set in the ATUMR, then the inbound message transactions are still logged to the Inbound message register but they do not block following vendor specific message transactions.

Table 117. Inbound Vendor_Defined Message Type 0 Response.

Response for Type 0 IVM	IVM Received Interrupt Mask (ATUIMR - bit 25)	Drop subsequent IVM (ATUCR - bit 6)
Unsupported Request (UR)	1	0
UR	1	1
Return UR when firmware requests a UR response. (PEMCSR -bit 14).	0	0
Dropped Silently when interrupt pending (ATUISR - bit 25).	0	1

Note: A Type0 vendor_defined message may be discarded without a UR response when the interrupt mask is cleared and the ATU is configured to drop subsequent IVM messages when the interrupt is pending.



3.3.2 Outbound Transactions

Outbound transactions initiated by the 81341 and 81342 core processor are directed to the PCI Express interface through the ATU. As a PCI Express requester, the ATU is capable of memory, I/O, configuration, and message transactions. Outbound memory transactions with addresses below 4GB use the short address format (32-bit address). Addresses above 4GB use the long address format (64-bit).

Outbound transactions use a separate set of queues from inbound transactions. Outbound write operations have their address entered into the outbound posted header queue (OPHQ) and their data into the outbound posted data queue (OPDQ). Outbound read transactions, use the Outbound Non-Posted Queue (ONPQ) to store address, and get data returned into the Inbound Completion Data Queue (ICPLDQ). Refer to [Section 3.8.2](#) for details of outbound queue architecture. Outbound configuration transactions use a special outbound port structure and are enqueue in the ONPQ. Refer to [Section 3.3.3](#) for details.

For outbound write transactions, the ATU is a target on the internal bus and a requester on the PCI Express Link. For outbound read transactions, the ATU is a completer on the internal bus (initially accepts the split read as a target and then provides read data by initiating a split completion). Internal bus operation is defined in [Section 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#).

Note: For all outbound writes, the byte enables must be contiguous. This means that write coalescing must be disabled in the Intel XScale® microarchitecture for transactions that target the outbound memory windows.

While Outbound I/O transactions are supported in all configurations, they should only be used when operating as the root complex. The PCI Express Specification states that PCI Express Endpoints must not generate I/O requests.



3.3.2.1 Outbound Address Translation - Internal Bus Transactions

In addition to providing the mechanism for inbound translation, the ATU translates Intel XScale® processor-initiated cycles to the PCI Express domain. This is known as *outbound address translation*. Outbound transactions are processor or ADMA transactions targeted at the PCI Express Link. The ATU internal bus target interface claims internal bus cycles and completes the cycle on the PCI Express Link on behalf of the Intel XScale® processor or DMAs.

Figure 27 shows 4 Gbyte memory section 0 (Internal Bus Address [35:32] = 0000₂) of the 81341 and 81342 memory map with all reserved address locations highlighted. The 64KByte outbound I/O window is from 0.FFFD.0000H to 0.FFFD.FFFFH while the PMMR registers reside from 0.FFD8.0000H to 0.FFDF.FFFFH.

By default, Outbound Memory Window 0, Outbound Memory Window 1, Outbound Memory Window 2, and Outbound Memory Window 3 reside in 4 Gbyte memory sections 1, 2, 3, and 4 respectively, of the 64 Gbyte Internal Bus address space.

The response of the ATU to Outbound Transactions is globally controlled by the Outbound Enable bit in the ATU Configuration Register as well as the Bus Master Enable bit in each function. When the Outbound Enable bit is deasserted, outbound transaction master-abort on the internal bus and are not forwarded to the PCI Express Domain. When the Outbound Enable bit is asserted, the relevant Bus Master Enable bit for each function is used to determine the appropriate response to an outbound transaction.

The Outbound ATUs behavior for the different combinations of these control bits is described in Table 118.

Table 118. Outbound Address Translation Control

Outbound Response	Outbound Enable ^a (ATUCR[1])	Bus Master Enable ^b
Master-Abort	0	0
Master-Abort	0	1
Retry	1	0
Claim ^c	1	1

a. In addition, the outbound memory windows need to be individually enabled in order to claim the transaction. When the memory window is disabled, it does not claim a transaction which might result in a Master-Abort. By default, Outbound Memory Windows 0 and 1 are enabled while Outbound Memory Windows 2, and 3 are disabled.

b. In a multi-function configuration, each function independently controls its own Bus Master Enable bit.

c. The ATU may respond with a Retry in this case when the Outbound Transaction Queues are full.



3.3.2.2 Outbound Address Translation Windows

Inbound translation involves a programmable inbound translation window consisting of a base and limit register and a value register for PCI to internal bus translation. The outbound address translation windows use a similar methodology except that the outbound translation window limit sizes are fixed in 81341 and 81342 internal bus address space; this removes the need for separate limit registers.

Figure 28 on page 254 illustrates the five outbound address translation windows. The ATU has four 4 Gbyte outbound memory translation windows and one 64 Kbyte outbound I/O translation window. By default, Outbound Memory Window 0 (OUMBAR0), Outbound Memory Window 1 (OUMBAR1), Outbound Memory Window 2 (OUMBAR2), and the Outbound Memory Window 3 (OUMBAR3) reside in 4 Gbyte memory sections 1, 2, 3, and 4, respectively. The default location of the 64 Kbyte outbound I/O window range is from 0.FFFD.0000H to 0.FFFD.FFFFH. The following registers are used to specify the five 4 Gbyte windows for claiming Outbound Memory transactions:

- Outbound Upper Memory Base Address Register 0 (OUMBAR0)
 - Default Value equal to 01H.
- Outbound Upper Memory Base Address Register 1 (OUMBAR1)
 - Default Value equal to 02H.
- Outbound Upper Memory Base Address Register 2 (OUMBAR2)
 - Default Value equal to 03H.
- Outbound Upper Memory Base Address Register 3 (OUMBAR3)
 - Default Value equal to 04H.
- Outbound I/O Base Address Register (OIOBAR)
 - Default Value equal to 0FFF D000H

An internal bus cycle with an address within one of the outbound windows initiates a read or write request on the PCI Express Link. The PCI transaction type depends on which translation window the local bus cycle “hits”. The read or write decision is based on the internal bus cycle type.

ATU has windows dedicated to the following outbound transaction types:

- Memory reads and Memory writes - Memory Window
- I/O reads and writes - I/O Window

Table 119. Internal Bus-to-PCI Command Translation for Memory Windows

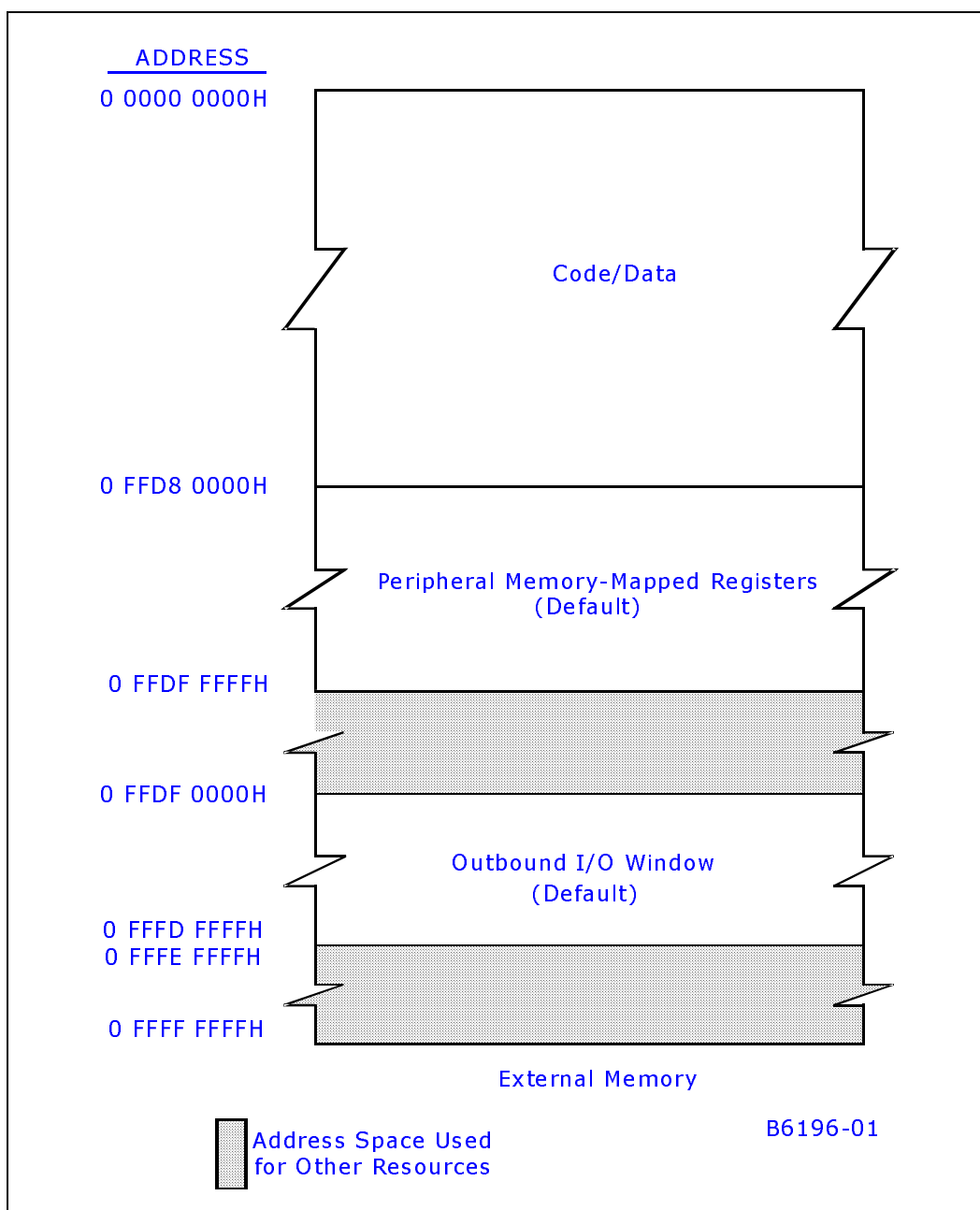
Internal Bus Command	PCI Express Transaction Type
Write	Memory Write Request
Read	Memory Read Request

Table 120. Internal Bus-to-PCI Command Translation for I/O Window

Internal Bus Command ^a	Conventional PCI Command
Write	I/O Write Request
Read	I/O Read Request

a. User should designate memory region containing I/O Window as non-cachable and non-bufferable from Intel XScale® processor. This insures all load/stores to I/O Window are of DWORD quantities. In the event that the user inadvertently issues a read to the I/O Window which crosses a DWORD address boundary, the ATU target aborts the transaction. Only bytes 3:0 is relevant dependent on the Byte Enables.

Figure 27. 4 Gbyte Section 0 of the Internal Bus Memory Map





The translation portion of outbound ATU transactions is accomplished with a value register in the same manner as inbound translations. Each outbound memory window is associated with one translation register which provides the upper translation addresses (OUMWVR0-3). When the corresponding OUMWVRx register is all-zero a 3DW header transaction is generated on the PCI Express Link. Otherwise, a 4DW header is generated on the PCI Express Link using the value in the OUMWVRx register for the upper 32-bit address. ATU uses the following registers during outbound address translation:

- Outbound Upper 32-bit Memory Window Value Register 0 (OUMWVR0)
- Outbound Upper 32-bit Memory Window Value Register 1 (OUMWVR1)
- Outbound Upper 32-bit Memory Window Value Register 2 (OUMWVR2)
- Outbound Upper 32-bit Memory Window Value Register 3 (OUMWVR3)
- Outbound I/O Window Value Register (OIOWVR)
- Outbound Configuration Cycle Address Register (OCCAR)

See [Section 3.16](#) for details on outbound translation register definition and programming constraints.

The translation algorithm used, as stated, is very similar to inbound translation. For memory transactions, the algorithm is:

Equation 10. Outbound Address Translation

$$\text{PCI Address} = (\text{Internal\_Bus\_Address} \& 0.\text{FFFF.FFFFH}) \mid (\text{Upper\_Window\_Value\_Register} \ll 32)$$

For memory transactions, the internal bus address is bitwise ANDed with the inverse of 4 Gbytes which clears the upper 4 bits of the 36 bit address. The result is bitwise ORed with the outbound upper window value register left shifted by 32 to create the Upper 32-bits of the PCI address. When the Upper 32-bits of the PCI Address equals 0000 0000H, the ATU generates a transaction with a 3DW header on the PCI Express Link, otherwise, a 4DW header is used.

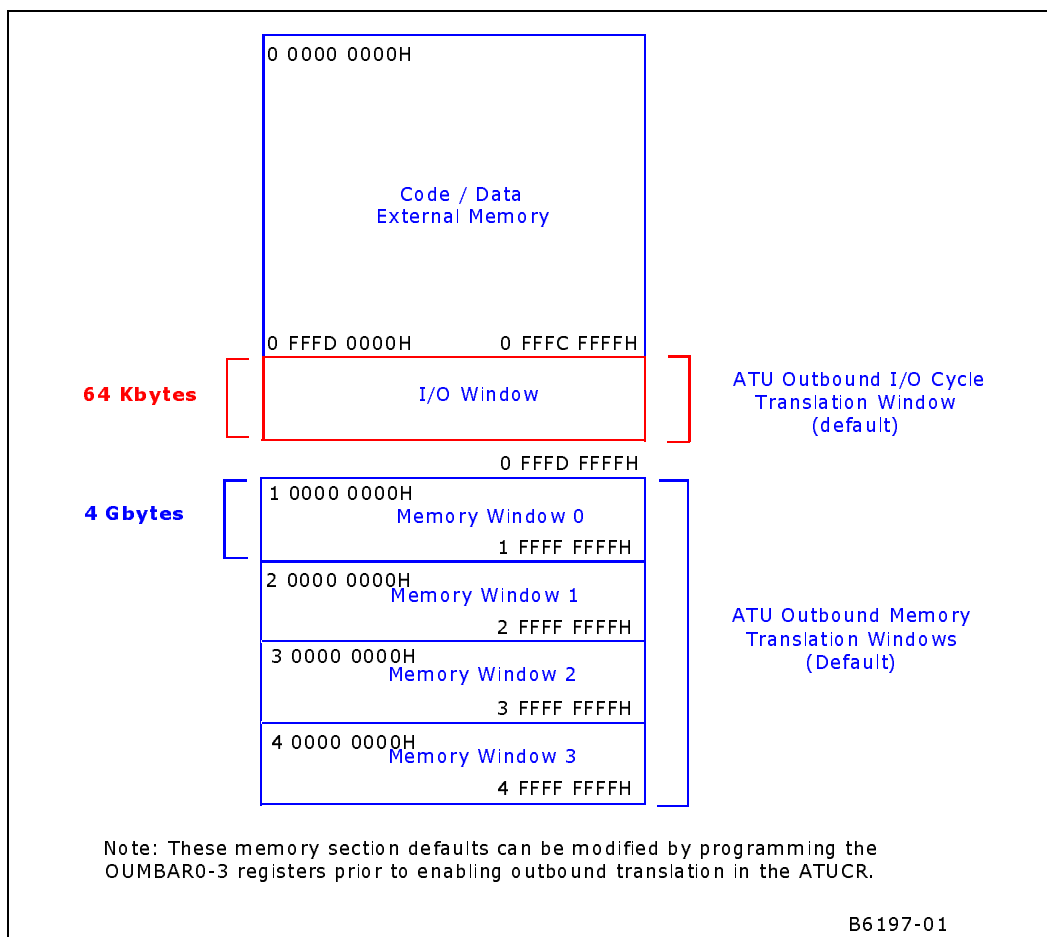
For I/O transactions, the algorithm is:

Equation 11. I/O Transactions

$$\text{PCI Address} = (\text{Internal\_Bus\_Address} \& 0.0000.\text{FFFFH}) \mid \text{Window\_Value\_Register}$$

For I/O transactions, the internal bus address is bitwise ANDed with the inverse of 64 Kbytes which clears the upper 20 bits of address. Address aliasing is prevented by the outbound window value registers which only allow values on boundaries equivalent to the window's length.

Figure 28. Outbound Address Translation Windows



3.3.2.3 Outbound DMA Transactions

The ATU provides all ADMA channels with a transparent path through the PCI Express interface. The entire 64-bit Host I/O Interface address programmed in the DMA descriptor is passed to the PCI Express link unmodified.

3.3.2.4 Outbound Function Number

The 81341 and 81342 is a multi-function device and the ATU associates each transaction with the correct PCI Express function number according to the following algorithm:

- Outbound Configuration transactions use the function number field specified in the “[Outbound Configuration Cycle Function Number - OCCFN](#)” on page 388.
- Outbound I/O transactions use the function number field specified in the “[Outbound I/O Base Address Register - OIOBAR](#)” on page 376.
- Outbound transactions targeting the outbound memory windows utilize the “Outbound Window x Function Number Mapping” programmed into bits 30:28 of the associated OUMBARx registers
- Outbound ADMA transactions use the “Host I/O Interface Function Number” programmed in the ADMA byte count register.



3.3.3 Outbound Write Transaction

An outbound write transaction is initiated by the Intel XScale® processor⁷ or by one of the DMAs and is targeted at a PCI Express domain. The outbound write address and write data are propagated from the 81341 and 81342 internal bus to a PCI Express Link through the OPHQ and OPDQ, respectively.

The ATUs internal bus target interface claims the write transaction and forwards it to the PCI Express Link. The data flow for an outbound write transaction on the internal bus is summarized in the following statements:

- ATU internal bus target interface latches the address from the internal bus into the OPHQ when that address is inside one of the outbound translate windows (see [Section 3.8](#)) and the OPHQ is not full.
- When the OPHQ is full, the target interface signals a Retry on the internal bus to the outbound cycle initiator.
- Once outbound address is latched, internal bus target interface stores write data into the OPDQ until the internal bus transaction completes or the reaches a buffer boundary.
- When the data is latched in a buffer in OPDQ, the outbound cycle is enabled for transmission on the PCI Express Link.

The PCI interface is responsible for completing the outbound write transaction with the PCI address translated from the OPHQ and the data in the OPDQ. The data flow for an outbound write transaction on the PCI Express Link is summarized in the following statements:

- Writes transactions is fragmented based on the Max_Payload_Size parameter. A write issues when the Max_Payload_Size is reached, or the write disconnects on the internal bus.
- When Posted Header and Posted Data credits are available a memory write request TLP is issued on the PCI Express Link.
- When a data parity error is detected while pulling data from the OPDQ, the TLP is poisoned.

7. For best performance, the user should designate the two Outbound Memory Windows as non-cachable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 126, "ATU Outbound Data Flow Ordering Rules" on page 269](#). In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cachable/non-bufferable and enforce the requirement in software.

3.3.4 Outbound Read Transaction

An outbound read transaction is initiated by the Intel XScale® processor⁸ or one of the DMAs and is targeted at a PCI slave on the PCI Express Link. The read transaction is propagated through the outbound non posted queue (ONPQ) and read data is returned through the inbound completion data queue (ICPLDQ).

The ATUs internal bus target interface claims the Memory Read transaction and forwards the read request through to the PCI Express Link and returns the read data to the internal bus.

The data flow for an outbound read transaction on the internal bus is summarized in the following statements:

- The ATU internal bus interface latches the internal bus address when the address is inside an outbound address translation window (or the direct addressing window, when enabled) and the ONPQ is not full. All read transactions are handled as split transactions. When the ONPQ is full (previous outbound transactions in progress), the internal bus interface signals a Retry to the transaction initiator.
- Read requests is fragmented into sub-requests based on the Max_Read_Request_Limit.
- When NPH credits are available, the ATU issue the read request when the head of the ONPQ has at least one entry and the ordering rules are satisfied.
- Once the request is issued, the Transaction Pending bit is set in the "PCI Express Device Status Register PE_DSTS".
- When a Completion with Completion Status of UR or CA is encountered, a flag is set and the ATU aborts the completion to the internal bus requester. The ONPQ is cleared of the transaction.
- Completions for subsequent sub-request that are already issued is marked for deletion and dropped once the completion returns.
- Once the transaction completes on the PCI Express Link, the ATU generates a completion transaction to return data to the internal bus requester.
- Once all outstanding request are satisfied, the Transaction Pending bit is cleared in the "PCI Express Device Status Register PE_DSTS"

8. For best performance, the user should designate the two Outbound Memory Windows as non-cachable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 126, "ATU Outbound Data Flow Ordering Rules" on page 269](#). In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cachable/non-bufferable and enforce the requirement in software.



3.3.5 Outbound Configuration Cycle Translation

The outbound ATU provides a port programming model for outbound configuration cycles.

Performing an outbound configuration cycle to the PCI Express Link involves up to two internal bus cycles:

1. Writing Outbound Configuration Cycle Address Register (OCCAR) with the bus, device, function, and register number used during the configuration cycle. The value of this register directly maps to bytes 8-11 of the configuration transaction header. See [Section 3.16.44, "PCI Express Message Control/Status Register - PEMCSR" on page 337](#) for information regarding configuration address cycle formats. This IB bus cycle enables the transaction.
2. Writing or reading the Outbound Configuration Cycle Data Register (OCCDR). A read causes a configuration cycle read to the PCI Express Link with the address in the outbound configuration cycle address register. Note that the Internal Bus read is executed as a split transaction. Similarly, a write initiates a configuration cycle write to PCI with the write data from the second processor cycle. Configuration cycles are non-burst and restricted to a single 32-bit word cycle⁹. This IB bus cycle executes the transaction.

When the Configuration Cycle Data Register is written, the data is latched and forwarded to the PCI Express Link. The Configuration Request TLP always uses function 0.

Note: Outbound configuration cycle data registers are not physical registers. They are a 81341 and 81342 memory mapped addresses used to initiate a transaction with the address in the associated address register. When the data register is accessed, the address is pulled from the ["Outbound Configuration Cycle Address Register - OCCAR"](#) to generate the TLP header and any write data is placed directly in the ONPDQ.

3.3.5.1 Outbound Configuration Cycle Error Conditions

When issuing configuration requests, the ATU must deal with receiving completions with Unsupported Request (UR) and Completer Abort (CA) status. When a UR or CA is received, the ATU interrupts the core by setting the Received Master Abort / Received Target Abort status in the ["ATU Interrupt Status Register - ATUISR" on page 333](#). The read cycle is terminated with a DABORT on the internal bus.

When the completion is returned with poisoned data, the ATU sets the Detected Parity Error Interrupt status bit in the ["ATU Interrupt Status Register - ATUISR" on page 333](#). The data is issued on the internal bus with bad parity.

3.3.5.2 Outbound Configuration Completions with Retry Status (CRS)

When issuing configuration requests, the ATU must deal with receiving a Configuration Request Retry Status (CRS). When a CRS is received, the ATU interrupts the core by setting the Received Configuration Retry Status in the ["ATU Interrupt Status Register - ATUISR" on page 333](#). A configuration read that is completed with a CRS also results in a DABORT on the internal bus.

It is the responsibility of the software to reissue the configuration transaction.

9. The user should designate the memory region containing the OCCDR as non-cachable and non-bufferable from the Intel XScale® processor. This insures that all load/stores to the OCCDR are only of DWORD quantities. In event the user inadvertently issues a read to the OCCDR that crosses a DWORD address boundary, the ATU target aborts the transaction. All writes are terminated with a Single-Phase-Disconnect and only bytes 3:0 are relevant.



3.3.5.3 Outbound PCI Express* Message Transactions

The ATU provides a port programming model to generate outbound PCI Express messages.

Generating an outbound message transaction to the PCI Express interface involves up to 5 internal bus cycles.

1. Write outbound message transaction header registers 0 - 3.
2. Write the data to the outbound message transaction payload register. This write causes the generation of the message TLP on the PCI Express interface.

Note: When the payload length is 0, a write to the payload register is still required but the written data is ignored and not sent as part of the message TLP.

3.3.5.4 Completion Timeout Mechanism

The ATU implements a completion timeout mechanism on all outbound requests that require completions. The timeout mechanism is fixed and ensures an expiration time between 16 and 32 ms.



3.4 Big Endian Byte Swapping

Each memory and I/O window has an associated byte swapping enable located in the following address translation registers:

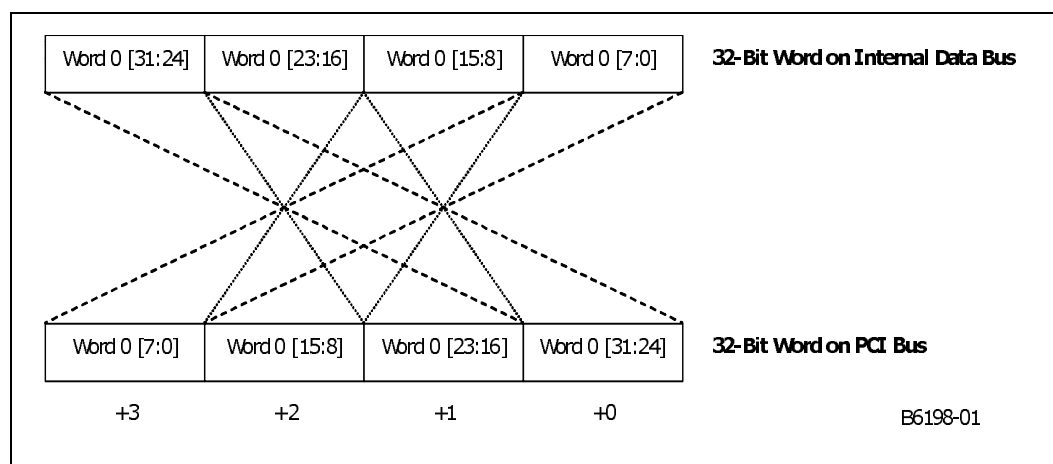
- bit 0 of Inbound Address Translate Value Register 0-2 (IATVR0-2)
- bit 0 of Inbound Expansion ROM Translate Value Register (ERTVR)
- bit 0 of Outbound I/O Window Translate Value Register (OIOWTVR)
- bit 27 of Outbound Upper Memory BAR 0-3 (OUMBAR0-3)

Note: The Messaging Unit (MU) Memory is mapped in PCI Window 0 (ATU Base Address Register 0) along with the MSI-X table structures. Byte swapping should not be enabled for BAR0 when using MSI-X.

3.4.1 Inbound Byte Swapping

When enabled, the swapping occurs as described in [Figure 29, “Inbound Byte Swapping” on page 259](#). The bytes are swapped within a DWORD and byte swapping is performed for all transactions regardless of byte count.

Figure 29. Inbound Byte Swapping



3.4.2 Outbound Byte Swapping

When enabled, the swapping occurs as described in Figure 30, “Outbound Byte Swapping for Transaction with Byte Count of 1” on page 260, Figure 31, “Outbound Byte Swapping for Transactions with Byte Count of 2” on page 260, and Figure 32, “Outbound Byte Swapping Transaction with Byte Count of 3 or Larger” on page 260. The bytes are swapped within a 32-bit DWORD and the type of byte swapping performed is determined by the transaction byte count. For Byte Count of 3 or larger transactions, no byte swapping is performed.

Note: The byte swapping capability of the ADMA unit should be used to swap bytes within each DWORD for PCI-to-Memory Read/Write DMA transfers.

Figure 30. Outbound Byte Swapping for Transaction with Byte Count of 1

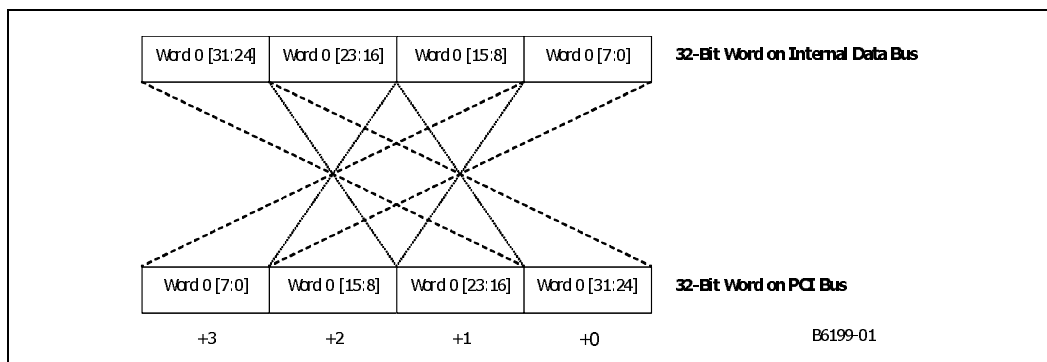


Figure 31. Outbound Byte Swapping for Transactions with Byte Count of 2

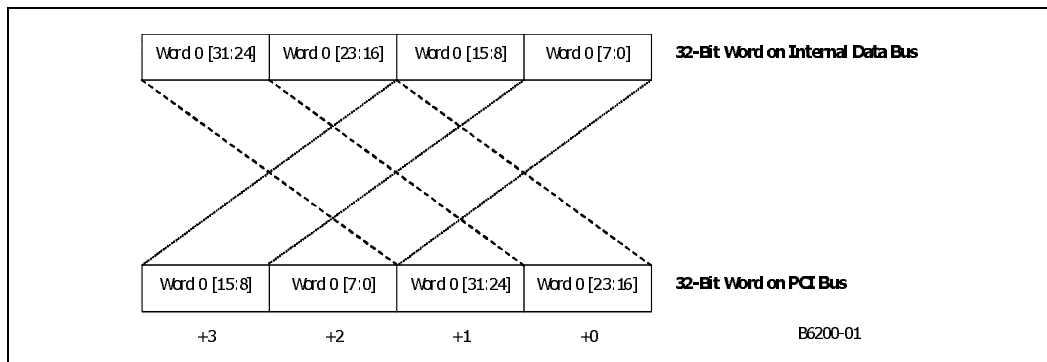
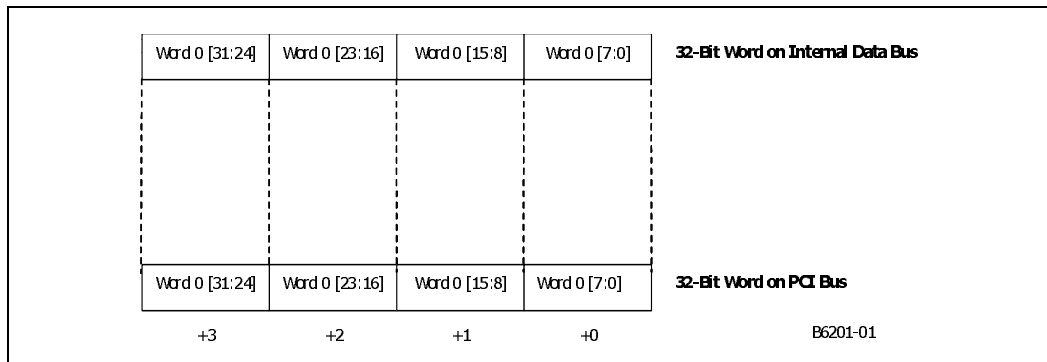


Figure 32. Outbound Byte Swapping Transaction with Byte Count of 3 or Larger





3.5 Messaging Unit

The Messaging Unit (MU) is used to transfer data between the PCI system and the 81341 and 81342 and notifies the respective system when new data arrives. The MU is located on the south internal bus of the 81341 and 81342 and is accessed via the ATU. The MU is described in [Chapter 4.0, "Messaging Unit"](#).

The PCI window for messaging transactions is the 8 Kbytes of the inbound translation window defined by the Inbound ATU Base Address Register 0 (IABAR0) and the Inbound ATU Limit Register 0 (IALR0).



3.6 PCI Express Messages

PCI Express defines a new message mechanism that is used to communicate outside of the normal Memory, I/O, and Configuration Spaces. The messages received and initiated by 81341 and 81342 vary depending on whether the device is acting as a root complex or an endpoint. All the messages defined in the PCI Express* specification and 81341 and 81342's actions are listed in [Table 121](#).

Table 121. Supported Message Types (Sheet 1 of 2)

Message	Description/ Comments	RC or End Point	Comment
Standard Messages initiated by SRL			
ERR_COR	Signal detection of a correctable error to Root Complex	RC, End Point	Set appropriate bit in ATUISR RC: Also log in PCI Express Error Source Identification Register
ERR_NONFATAL	Signal detection of an uncorrectable error Root Complex		
ERR_FATAL	Signal detection of a fatal error Root Complex		
PM_Active_State_NAK	Rejection of request to enter a low power state from an End Point or Upstream Port.	RC	
PME_Turn_Off ^a	Notification of pending turn-off of link clock and power	RC	Generated from PEMCSR.
PM_PME ^b	PME message conveying the ID of the PME originator	End Point	
PME_TO_Ack	Acknowledge turn-off of link clock and power	End Point	Automatically generated on receipt of PME_Turn_Off message
Assert_INTx	Assert INTx virtual signal	End Point	Generate to reflect signals from Interrupt Controller.
Deassert_INTx	De-assert INTx virtual signal	End Point	
Set_Slot_Power_Limit	Set Slot Power Limit in Upstream Port	RC	Generated due to write to Slot Capabilities register or anytime Link transitions from non-DL_Up status to a DL_Up status.
Vendor Defined Message Type 0/1	Vendor Specific Message. It is used by devices to communicate with SRL device core.	RC, End Point	Generated from Outbound Vendor Message register.
Attention_Indicator_X	These messages are issued by the root port to change the indicator on the device	RC	Generate due to change in indicator control bits in Slot Control Register
Power_Indicator_X	These messages are issued by the root port to change the indicator on the device	RC	
Attention_Button_Pressed	Issued by endpoint when button pressed on device	End Point	Generated from PEMCSR

**Table 121. Supported Message Types (Sheet 2 of 2)**

Message	Description/ Comments	RC or End Point	Comment
Standard Messages accepted by SRL as target			
ERR_COR	Signal detection of a correctable error to Root Complex	RC	Log in PCI Express Advanced Error registers and interrupt the core via the root interrupt bit in the ATUISR
ERR_NONFATAL	Signal detection of an uncorrectable error Root Complex	RC	
ERR_FATAL	Signal detection of a fatal error Root Complex	RC	
PM_Active_State_NAK	Rejection of request to enter a low power state	End Point	
PME_Turn_Off	Notification of pending turn-off of link clock and power	End Point	Generate PME_TO_Ack message.
PM_PME	PME message conveying the ID of the PME originator	RC	Log in requester ID in the Root Status Register. Interrupt the core via the Root Complex interrupt bit in the ATUISR
PME_TO_Ack ^c	Acknowledge turn-off of link clock and power	RC	
Assert_INTx	Assert INTx virtual signal	RC	Captured and converted to signals to Interrupt Controller
Deassert_INTx	De-assert INTx virtual signal	RC	
Set_Slot_Power_Limit	Set Slot Power Limit in Upstream Port	End Point	Copy payload to Device Capabilities register and interrupt the core.
Vendor Defined Message Type 0/1	Vendor Specific Message. It is used by devices to communicate with SRL device core.	RC, End Point	Log in Inbound Vendor Message Register and interrupt the core.
Attention_Indicator_X	These messages are issued by the root port to change the indicator on the device	End Point	Log in the PEMCSR, and interrupt the core
Power_Indicator_X	These messages are issued by the root port to change the indicator on the device	End Point	Log in the PEMCSR, and interrupt the core
Attention_Button_Pressed	Issued by endpoint when button pressed on device	RC	Log in Slot Status Register and interrupt core

- As a Root Complex, system software never requests SRL to broadcast the PME_Turn_Off message in preparation for removing the main power and clock sources. Thus, SRL doesn't generate this message as a RC. However, for reuse purpose the Message Unit is capable of generating this message.
- As an End Point, SRL never generates a PM_PME message to the RC. It means SRL doesn't consume AUX power. However, for reuse purpose the Message Unit is capable of generating this message.
- SRL doesn't generate PME_Turn_Off message. Thus it shouldn't receive PME_TO_Ack Message. However, for reuse purpose the Message Unit is capable of accepting this message.



3.7 Expansion ROM Translation Unit

The inbound ATU supports one address range (defined by a base/limit register pair) used for the Expansion ROM. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details on Expansion ROM format and usage.

During a powerup sequence, initialization code from Expansion ROM is executed once by the host processor to initialize the associated device. The code can be discarded once executed. Expansion ROM registers are described in [Section 3.16.22](#).

The inbound ATU supports an inbound Expansion ROM window which works like the inbound translation window. A read from the expansion ROM windows is forwarded to the internal bus. The address translation algorithm is the same as the inbound translation; see [Section 3.3.1.1, "Inbound Address Translation"](#) on page 242.

The Expansion ROM unit uses the ATU inbound transaction queue and the inbound read data queue.

Expansion ROM writes are not supported and result in a Completer Abort.

Note: Both the Memory Space enable and Expansion ROM Base Address Enable bits must be set to 1 before the ATU accepts accesses to its Expansion ROM.



3.8 ATU Queue Architecture

ATU operation and performance depends on queueing mechanism implemented between internal bus interface and PCI Express interface. [Figure 24](#) indicates the ATU queue architecture consists of separate inbound and outbound queues. The function of each queue is described in the following sections.

3.8.1 Inbound Queues

The inbound data queues of the ATU support transactions initiated on a PCI Express Link and targeted at either 81341 and 81342 local memory or a 81341 and 81342 memory mapped register. [Table 122](#) details the name and sizes of the ATU inbound data queues.

Table 122. Inbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
IPDQ	Inbound Posted Data Queue	3.75 KBytes (240 credits)
IPHQ	Inbound Posted Header Queue	16 Headers
INPQ	Inbound Non Posted Queue	8 Headers ^a
ICPLDQ	Inbound Completion Data Queue	4 KBytes (infinite credits) ^b
ICPLHQ	Inbound Completion Header Queue	4 Headers (infinite)

- a. Non Posted request with data (I/O and Configuration Writes) always use the 3DW header. The associated data is always 1 DW in size and can be stored the 4th DW of the header queue.
- b. As a PCI Express endpoint the ATU must pre-allocate buffer space before issuing a read request. The ATU is required to advertise infinite credits.

3.8.1.1 Inbound Posted Queue Structure

The ATU Inbound Posted Queues consist of the inbound posted data queue (IPDQ) and the inbound posted header queue (IPHQ). The inbound posted data queue holds the data for posted (memory/message) transactions moving from a PCI Express Link to the internal bus and the header queue holds the corresponding address. The inbound posted data queue has a queue depth of 4 KBytes and moves posted transactions from the PCI Express Link to the internal bus. The corresponding header queue, IPHQ, is capable of holding 8 entries.

The following rules apply to the PCI Express Link interface and govern the acceptance of data into inbound posted queues:

- Posted transactions are drained from the head of the queue when the master interface has acquired bus ownership and transaction ordering and priority have been satisfied (see [Section 3.8.3](#)). A memory write transaction is considered drained from the queue when the entire amount of data entered on the PCI Express Link has been accepted by the internal bus target. Error conditions resulting in the cancellation of a write transaction only flush the transaction at the head of the data and address queue. All other transactions within the queues are considered still valid.



3.8.1.2 Inbound Non Posted Queue Structure

The inbound read queues are responsible for retrieving data from local memory and returning it to the PCI Express Link in response to a read transaction initiated from a PCI master. Up to 8 non posted transactions can be held in the INPQ. The read data is returned through OCPLDQ.

For Configuration and I/O writes, both the header and data are placed in the INPQ. Configuration and I/O transactions always utilize the 3DW header and the data for the write transactions is always 1DW. So the data can be placed in the same queue as the header. The advantage to this is the NPD (non-posted data) credits can be advertised as infinite and the header credits prevents overrunning the INPQ.

Read requests are fragmented into 1K aligned requests before issuing to the internal bus/

3.8.1.3 Inbound Completion Queue Structure

The inbound completion queue provides insures space for all outstanding outbound read requests. This queue is 4KB in size and is used to order the completion data before returning it to the internal bus. When an outbound internal bus request is fragmented into multiple PCI Express request this queue ensures that the data is returned in order to the internal requesting agent.

3.8.1.4 Inbound Transaction Queues Command Translation Summary

Table 123. PCI to Internal Bus Command Translation for All Inbound Transactions

PCI Express TLP	Internal Bus Command
Memory Read	Read
Memory Read - Locked	none - Unsupported Request
Memory Write	Write
I/O Read	Read
I/O Write	Write
Configuration Read Type 0	Read
Configuration Write Type 0	Write
Configuration Read Type 1	none - Unsupported Request
Configuration Write Type 1	none - Unsupported Request
Message	none - Handled by express message unit
Message with Data	none - Handled by EMU
Message Advanced Switch	none - Unsupported Request
Message Advanced Switch with Data	none - Unsupported Request
Completion without Data	
Completion with Data	
Completion - Locked without Data (error condition)	none
Completion - Locked with Data	
Others (reserved encodings)	



3.8.2 Outbound Queues

The outbound queues of the ATU are used to hold read and write transactions from the core processor directed at the PCI Express Link. Each ATU outbound queue structure has a separate read queue, write queue, and address queue. [Table 124](#) contains information about ATU outbound queues.

Table 124. Outbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
OPDQ	Outbound Posted Data Queue	4 KBytes
OIPHQ	Outbound Posted Header Queue	16 Headers
ONPQ	Outbound Non Posted Queue	8 Headers ^a
OCPLDQ	Outbound Completion Data Queue	4 KBytes
OCPLHQ	Outbound Completion Header Queue	4 Headers

a. Non Posted request with data (I/O and Configuration Writes) always use the 3DW header. The associated data is always 1 DW in size and can be stored the 4th DW of the header queue.

The outbound queues are capable of holding outbound memory read, memory write, I/O read, and I/O write transactions. The type of transaction used is defined by the internal bus address and the command used on the internal bus. See [Section 3.3.2](#) and [Section 3.3.3](#) for details on outbound address translation.

When an internal bus agent initiates an outbound write transaction, the address is entered into the OWADQ (when not full). The data from the internal bus write is then entered into the OWQ and the transaction is forwarded to the PCI Express Link. When the write completes (or an error occurs), the address is flushed from the OWADQ. Data is flushed only for the master abort or target abort cases.

For outbound reads, the address is entered into the OTQ (when not full) and a split response termination is signaled to the requester on the internal bus. Read data is fetched and returned to the requester on the internal bus.

3.8.2.1 Relaxed Ordering and No Snoop Outbound Request Attributes

The ATU may set the Relaxed Ordering (RO) and/or the No Snoop (NS) bits for an outbound request.

For outbound Memory Read and Memory Write requests initiated by the ADMA the values of the attribute bits are controlled by the ADMA Descriptor Control Word.

For any other outbound requests, the NS and RO attribute bits is set to '0'.

Note: When the Enable Relaxed Ordering or Enable No Snoop bits are cleared in the "PCI Express Device Control Register - PE_DCTL" on [page 348](#), then the ATU forces the RO and NS attributes to '0' respectively for all transactions.



3.8.3 Transaction Ordering

Because the ATU can process multiple transactions, they must maintain proper ordering to avoid deadlock conditions and improve throughput. The ATU transaction ordering rules used by the 81341 and 81342 are listed in [Table 125](#) for the inbound direction and [Table 126 on page 269](#) for the outbound direction. The tables are based on the direction the transaction is moving, i.e. the data for inbound completion moves in the same direction as the data for an inbound write or the address/command for an inbound read.

Note: Outbound Non-Posted Writes are the result of Internal Bus Memory writes that are claimed by either the I/O translation window or the [Outbound Configuration Cycle Data Register - OCCDR](#). Though these write requests arrive on the PCI Express Link as non-posted write requests, it is important to note that from the Intel XScale® processor's point of view, these internal bus memory write requests are posted into the Outbound ATU transaction queue. Thus, even though a write completion is returned to the ATU on the PCI Express Link for outbound non-posted write request, the write completion is not passed back through to the internal bus. Additionally, strong ordering between outbound memory (posted) write requests and outbound non-posted write requests are **not** maintained as indicated in [Table 126 on page 269](#).

For best performance, the user should designate the two Outbound Memory Windows as non-cachable and bufferable from the Intel XScale® processor. This assignment enables the Intel XScale® processor to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in [Table 126, "ATU Outbound Data Flow Ordering Rules" on page 269](#). In the event that the user requires strict ordering to be maintained, the user must change the designation of this region of memory to be non-cachable/non-bufferable and enforce the requirement in software.

Table 125. ATU Inbound Data Flow Ordering Rules

Row Pass Column?	Inbound Write or Message Request	Inbound Read Request	Inbound Configuration Write Request	Inbound Read Completion	Inbound Configuration or I/O Write Completion
Inbound Write or Message Request	No	Yes	Yes	Yes	Yes
Inbound Read Request	No	No	No	Yes	Yes
Inbound Configuration Write Request	No	No	No	Yes	Yes
Inbound Read Completion	No	Yes	Yes	Yes?	Yes?
Inbound Configuration or I/O Write Completion ^a	N/A	N/A	N/A	N/A	N/A

a. Inbound Completions for Configuration writes and I/O writes are not applicable since these transactions are never passed back to the Internal Bus Requester (Intel XScale® processor). The reason is that from the Intel XScale® processor's point of view, these write requests are posted into the Outbound ATU transaction queue.



Definitions of the terms used in Table 125 and Table 126 are as follows. PCI terms are noted in parenthesis:

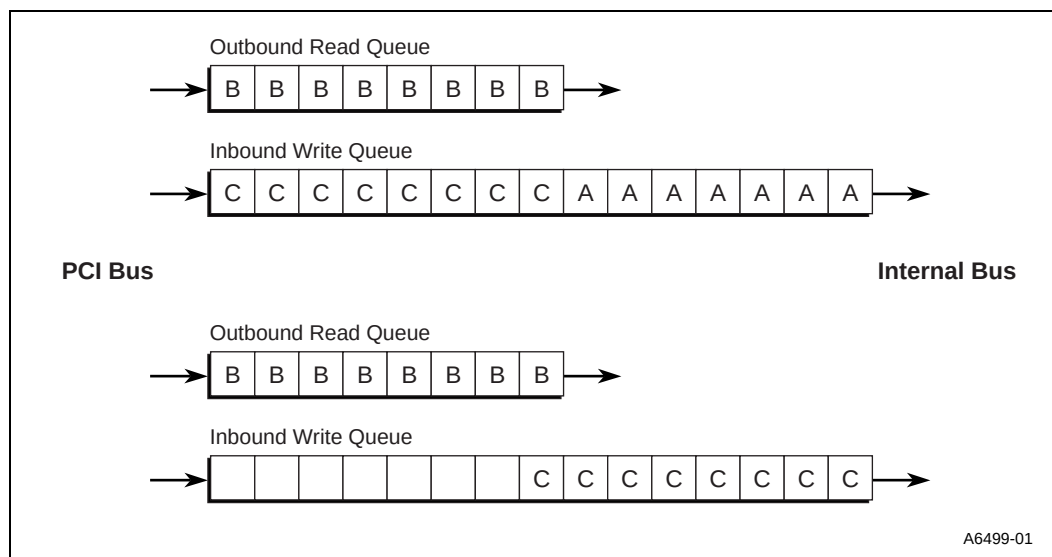
Table 126. ATU Outbound Data Flow Ordering Rules

Row Pass Column?	Outbound Write or Message Request	Outbound Read Request	Outbound Configuration Write Request	Outbound Read Completion	Outbound Configuration or I/O Write Completion
Outbound Write or Message Request	No	Yes	Yes	Yes	Yes
Outbound Read Request	No	No	No	Yes	Yes
Outbound Configuration Write Request	No	No	No	Yes	Yes
Outbound Read Completion	No	Yes	Yes	Yes	Yes
Outbound Configuration or I/O Write Completion	No	Yes	Yes	Yes	Yes

These transaction ordering rules define the way in which data moves in both directions through the ATU. In Table 125 and Table 126 a **NO** response in a box means that based on ordering rules, the current transaction (the row) can not pass the previous transaction (the column) under any circumstance. A **Yes** response in the box means that the current transaction is *allowed* to pass the previous transaction but is not required to, based on whether a consistent view of data or prevention of deadlocks are needed.

In the case of inbound write operations, multiple transactions may exist within the IPHQ and the corresponding IPDQ at any point in time. The ordering of these transactions is based on a time stamp basis. Transactions entering the queue are stamped with a relative time in relation to all other transactions moving in a similar direction.

Example 5. Inbound Queue Completion





In [Example 5 on page 269](#), the inbound write and outbound read queues of the ATU are shown. In this example, transaction A entered the write queue at **Time 0**. Next, the ATU entered read data into the outbound read queue at **Time 1** (Transaction B). Finally, before the previous transactions could be cleared, another inbound write, Transaction C, was entered into the IWQ. The ordering in [Table 125](#) states that nothing can pass an inbound write and therefore Transaction A must complete on the internal bus before Transaction B since an outbound read completion can not pass an inbound write. Also, Transaction A must complete before Transaction C since an inbound write can not pass another inbound write. Once Transaction A completes, Transaction C moves to the head of the IWQ. The two transactions at the head of the queues moving data in an inbound direction are now Transaction C, an inbound write, and Transaction B, an outbound read completion. Ordering states that an inbound write may pass an outbound read completion. This means that the arbitration mechanism now takes over to decide which completes. Note that ordering enforced the completion of Transaction A but arbitration dictated the completion of Transactions B and C.

The first action performed to determine which transaction is allowed to proceed (either inbound or outbound) is to apply the rules of ordering as defined in [Table 125](#) and [Table 126](#). Any box marked **No** must be satisfied first. For example, when an inbound read request is in ITQ and it was latched *after* the data in the IDWQ arrived (this is a configuration write), then ordering states that an Inbound Read Request may not pass an Inbound Configuration Write Request. Therefore, the Inbound Configuration Write Request must be cleared out of IDWQ before the Inbound Read Request is attempted on the internal bus. Once transaction ordering is satisfied, the boxes marked **Yes** are now resolved.



3.8.3.1 Transaction Ordering Summary

Table 127 and Table 128, define transaction ordering in relation to token assignment of the priority mechanism (see Section 3.8.3). These tables are read as follows:

1. As transaction enters the respective queue head, the question in column 2 is asked.
2. When all the answers in column 3 for a given transaction type assigns a token to the transaction at the head of the queue, a token is assigned. Otherwise, no token is assigned signifying that transaction ordering must first be satisfied. Any transaction with a token may be initiated on the bus.

Table 127. Inbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Inbound Posted Transaction Write in IPHQ	Is there an Inbound Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Non Posted Request in INPHQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Non Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Completion in ICPLHQ	Is the relaxed order bit set in the header and the Enable Relaxed Ordering bit set in the ATUCR?	Yes	Assign Relaxed Order Token
		No	Check for earlier Posted Request
	Is there an Inbound Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Completion with an earlier timestamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token

Table 128. Outbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Outbound Posted in OPHQ	Is there an Outbound Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Non Posted Request in ONPHQ	Is there an Outbound Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Outbound Non Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Completion in the OCPLHQ	Is there an Outbound Posted Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Outbound Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete
		No	Assign Token



3.8.4 Byte Parity Checking and Generation

The ATU internal bus interface supports byte-wise parity protection on the internal bus. This includes ADDP[4:0] and DATAP[15:0] on the address bus (A[35:0]) and the data bus (D[127:0]) respectively.

For an outbound request the ATU check the address parity before claiming the request on the internal bus. When an error occurs, the transaction is not claimed. The Data parity information is captured off the internal bus and stored in the internal queues. At the PCI Express interface, the parity information is checked as the data packet is transferred on the link. When a parity error occurs, the packet is nullified by using the End Bad marker and state is updated so corrective action can be taken when the packet is replayed on the PCI Express link. Header parity errors, which can only occur in the retry buffer, results in a malformed packet that is dropped by the component on the other side of the link. Data parity errors results in the EP bit being set to notify the target of the TLP that the data is corrupt.

For an inbound write request, the ATU computes and appends address parity and data parity before placing the TLP in the inbound queues. When an ECRC violation is detected the packet is treated as when an address parity error occurred and the entire packet is dropped without forwarding it to the internal bus. When a poisoned TLP is received the parity for the entire payload is inverted so that the internal bus target detects bad parity on all bytes.

3.8.4.1 Parity Generation

Data parity signals include byte enables in the calculation. Table 129 lists data bits that are used for parity calculation. Parity bits are calculated by bit XOR-ing the data bits as shown in Table 129. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is calculated as follows:

Equation 12. $\text{DATAP0} = \text{D}[0] \text{ XOR } \text{D}[1] \text{ XOR } \text{D}[2] \text{ XOR } \text{D}[3] \text{ XOR } \text{D}[4] \text{ XOR } \text{D}[5] \text{ XOR } \text{D}[6] \text{ XOR } \text{D}[7] \text{ XOR } \text{WBE}[0]$

Table 129. Parity Generation

Address/Data Parity Bit	Address/Data Bus	Address/Data Parity Bit	Address/Data Bus
ADDP4	A[35:32]	DATAP9	D[79:72], WBE[9]
ADDP3	A[31:24]	DATAP8	D[71:64], WBE[8]
ADDP2	A[23:16]	DATAP7	D[63:56], WBE[7]
ADDP1	A[15:8]	DATAP6	D[55:48], WBE[6]
ADDP0	A[7:0]	DATAP5	D[47:40], WBE[5]
DATAP15	D[127:120], WBE[15]	DATAP4	D[39:32], WBE[4]
DATAP14	D[119:112], WBE[14]	DATAP3	D[31:24], WBE[3]
DATAP13	D[111:104], WBE[13]	DATAP2	D[23:16], WBE[2]
DATAP12	D[103:96], WBE[12]	DATAP1	D[15:8], WBE[1]
DATAP11	D[95:88], WBE[11]	DATAP0	D[7:0], WBE[0]
DATAP10	D[87:80], WBE[10]		



3.8.4.2 Parity Checking

On an outbound request, address parity is checked on the address bus A[35:0]. The parity bits are checked by first bit XORing the address bits shown in Table 129 with the corresponding address parity bits, and then verifying when the result of each of the XORed operations are equal to zero. As an example, the parity calculation for the lowest order byte of the address bus A[7:0] is carried as follows:

Equation 13. $\text{PARITY\_RESULT} = \text{ADDP0} \text{ XOR } A[0] \text{ XOR } A[1] \text{ XOR } A[2] \text{ XOR } A[3] \text{ XOR } A[4] \text{ XOR } A[5] \text{ XOR } A[6] \text{ XOR } A[7]$

The parity logic uses the following algorithm. This algorithm logs the error when an error is detected.

```
check address parity
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

On an outbound request with data, data parity is checked on the data bus D[127:0]. The parity bits are checked by first bit XORing the data bits shown in Table 129 with the corresponding data parity bits, and then verifying when the result of each of the XORed operations is equal to zero. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is carried as follows:

Equation 14. $\text{PARITY\_RESULT} = \text{DATAPO} \text{ XOR } D[0] \text{ XOR } D[1] \text{ XOR } D[2] \text{ XOR } D[3] \text{ XOR } D[4] \text{ XOR } D[5] \text{ XOR } D[6] \text{ XOR } D[7] \text{ XOR } \text{WBE}[0]$

A non-zero result from the above operation indicates a parity error.

The parity logic uses the following algorithm, and this algorithm logs the error when an error is detected.

```
check data parity
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

3.8.4.3 Parity Disabled

When software disables parity, the ATU does generate the parity inbound transactions, but does not check the parity on outbound transactions.



3.9 ATU Error Conditions

PCI Express and internal bus error conditions cause ATU to log header information and set status bits to inform error handling code of exact cause of error condition. Two sets of registers are provided to allow independent control by both the Host processor and the internal Intel XScale® microarchitecture. Error conditions and status can be found in the ATUSR. The basic flow for a PCI Express error is as follows:

- Log the Error in the PCI Express Advanced Error and the PCI Interface Error registers
- Set the bit in the ATU Status Register which corresponds to the error condition (master abort, target abort, etc.)
- Set the bit in the ATU Interrupt Status Register which corresponds to the error condition (master abort, target abort, etc.). This function is maskable for all PCI error conditions.
- The setting of the bit in the ATU Interrupt Status Register results in an interrupt being driven to the Intel XScale® processor.

Error conditions on one side of the ATU are generally propagated to the other side of the ATU and have different effects depending on the error. Error conditions and their effects are described in the following sections.

PCI Express error conditions and the action taken on the link are defined within the *PCI Express Base Specification*, Revision 1.0a. The ATU adheres to the error conditions defined within the PCI specification for both requester and completer operation. Error conditions on the internal bus are caused by an ECC error from the Memory Controller, (see [Section 7.5, "ECC Interrupts/Error Conditions" on page 621](#) for details on memory controller error conditions), an Internal Bus Byte Parity Error, or by incorrect addressing resulting in an internal master abort or target abort. All actions on the PCI Express interface for error situations are dependent on the error control bits found in the ATU Command Register ([Section 3.16.5, "ATU Command Register - ATUCMD" on page 302](#)), the PCI Express Device Control Register ([Section 3.16.59, "PCI Express Device Control Register - PE_DCTL" on page 348](#)) and the PCI Express Advanced Error Masks ([Section 3.16.71, "PCI Express Uncorrectable Error Mask - ERRUNC_MSK" on page 361](#) and [Section 3.16.74, "PCI Express Correctable Error Mask - ERRCOR_MSK" on page 364](#)).

The following sections detail all ATU error conditions on the PCI Express and 81341 and 81342 internal bus, action taken on these conditions, and status and control bits associated with error handling.



3.9.1 PCI Express* Errors

PCI Express classifies errors as either Fatal, Uncorrectable or Correctable which allows the platform to map errors to a suitable handling mechanism. The control and mapping of errors into each of these categories are provided by the Advanced Error Handling registers. In addition to the specification defined registers, the ATU provides a duplicate set of error log in registers (see “[PCI Interface Error Control and Status Register - PIE_CSR](#)” on page 397) that allows the 81341 and 81342 component to respond to errors in an application specific fashion.

3.9.1.1 Role Based Error Reporting

In earlier versions of the PCI Express Specification, errors were reported by the agent that detected the error. The *PCI Express Base Specification*, Revision 1.1 implements a role based error reporting where the response to the errors is based on the components role in the transaction. In general, errors detected in Non Posted transactions are handled by the initial requestor and the completer may optionally send an advisory message to the root complex as an ERR_COR message. Errors in Posted transactions are still logged and reported by the target device.

Note: When the severity for the error is programmed to fatal in the [PCI Express Uncorrectable Error Severity - ERRUNC_SEV](#) register, then it is not an “Advisory Non-Fatal Error” and is signalled with an ERR_FATAL message. A fatal severity overrides all other Advisory Error control bits.

The following errors are considered “Advisory Non-Fatal Error” cases and have different handling depending based on the Transaction type.

- ECRC Check Failed
- Unexpected Completion
- Unsupported Request (UR)
- Poisoned TLP Received
- Completer Abort (CA)
- Completion Timeout

Table 130. Advisory Error Cases

Error Type	Posted	Non Posted	Completion
ECRC Check Failed	Not an Advisory Error - Send ERR_NONFATAL		
Unsupported Request	Not Advisory Error - Send ERR_NONFATAL	Advisory Error - Send ERR_COR	Signaled via device driver
Completer Abort	Not Advisory Error - Send ERR_NONFATAL	Advisory Error - Send ERR_COR	Signaled via device driver
Unexpected Completion	N/A	N/A	Advisory Error - Send ERR_COR
Poisoned TLP Received	When PIE_AEC bit 4 is set then treat as an Advisory Error - send ERR_COR. Else Not Advisory Error - Send ERR_NONFATAL	N/A	When PIE_AEC bit 5 is set then treat as an Advisory Error - send ERR_COR. Else Not Advisory Error - Send ERR_NONFATAL
Completion Timeout	N/A	N/A	When PIE_AEC bit 6 is set then treat as an Advisory Error - send ERR_COR. Else Not Advisory Error - Send ERR_NONFATAL

The different responses are described in detail in the following sections.



3.9.1.2 Malformed Packets

The following checks are made to detect malformed TLPs.

- Data Payload exceeds the length specified by the value in the Max_Payload_Size field of the Device Control Register.
- The value in the length field and the actual amount of data received do not match. The value in the length field applies only to data, TLP digest is not included in the length.
- A TLP with a 1b in TD field but without a TLP digest or a TLP with a TLP digest but without a 1b in TD field
- Address/Length combination which crosses a 4K boundary.
- When 81341 and 81342 is operating as Endpoint, and ATUE receives Assert_INTx/Deassert_INTx messages.
- Assert_INTx/Deassert_INTx messages do not use default Traffic Class (TC0)
- Power Management messages do not use default Traffic Class (TC0)
- Error Signalling Messages do not use default Traffic Class (TC0)
- Packets having undefined Type Field
- IO and Configuration requests are considered malformed when
 - TC[2:0] /= 000b
 - Attr[1:0] /= 00b
 - Length[9:0] /= 0000000001b
 - Last DW BE[3:0] /= 0000b
- For Read Completion, when length = 0 and the completion status /= 000, 001, 010,

When a malformed packet is detected, the packet is dropped and the error is logged. No flow control information is updated for malformed packets.

3.9.1.3 ECRC Check Failed

Return ERR_NONFATAL / ERR_FATAL depending on the severity setting. ERR_COR is never generated.

This component is never an "Intermediary Receiver" so the advisory error condition does not apply.



3.9.1.4 Unsupported Request

Unsupported Requests are detected by the address decode and translation logic. A TLP is treated as unsupported in the following cases:

- the TLP fails to match any of the active Memory or I/O windows.
- a configuration TLP that targets an invalid function number
- receipt of a Vendor_Defined Type 0 message and
 - the inbound vendor message received interrupt mask is set (ATUIMR[25])
 - the inbound vendor_defined type 0 UR response bit is set (PEMCSR[14])
- a message request with an undefined or unsupported Message Code
- a poisoned I/O or Configuration request
- receipt of a Memory or I/O transaction while in a non-D0 power state
- receipt of a Memory Read Lock (MRdLk)

No checks are made to for address + length crossing a window boundary.

For posted transactions, this is **not** an Advisory Error and an ERR_NONFATAL is sent to the root complex.

For non-posted transactions, this is considered and Advisory Error. An ERR_COR is sent to the root complex and a completion with UR status is returned to the requestor.

Note: When the severity setting in “[PCI Express Uncorrectable Error Severity - ERRUNC_SEV](#)” register is fatal this is not an Advisory Error and an ERR_FATAL is sent to the root complex.

3.9.1.5 Completer Abort

Requests that target abort or master abort on the Internal Bus are treated as a Completer Abort.

These requests must have passed the Malformed TLP checks as well as the Unsupported Request checks before they are issued on the internal bus.

For posted transactions, an ERR_NONFATAL is sent to the root complex.

For non-posted transactions, an ERR_COR is sent to the root complex and a completion with CA status is returned to the requestor.

Note: When the severity setting in “[PCI Express Uncorrectable Error Severity - ERRUNC_SEV](#)” register is fatal this is not an Advisory Error and an ERR_FATAL is sent to the root complex.

3.9.1.6 Unexpected Completions

Unexpected completions occur when a completion transaction ID does not match a outstanding request. When the Requestor ID of the completion matches a valid function, the error gets logged in that function. Otherwise the error gets logged against all functions.

This an Advisory Non-Fatal Error and an ERR_COR is sent to the root complex.

Note: When the severity setting in “[PCI Express Uncorrectable Error Severity - ERRUNC_SEV](#)” register is fatal this is not an Advisory Error and an ERR_FATAL is sent to the root complex.



3.9.1.7 Poisoned TLP Received

Poisoned TLPs can be received for both Inbound Posted (Write/Message) and Inbound Completions. The two TLP types can be handled differently.

Poisoned completions are passed through to the internal bus with bad parity. In addition to the Advanced Error requirements, the TLP header and DMA descriptor are logged in the PCI Interface Error Log (PIE_LOGx) registers and the Intel XScale® microarchitecture are interrupted. When bit 4 is set in the ["PCI Express Advisory Error Control Register - PIE_AEC"](#), this is treated as an "Advisory Error" and an ERR_COR is sent to the root complex; otherwise an ERR_NONFATAL is issued.

Poisoned Memory Writes are passed through to the internal bus with bad parity. In addition to the Advanced Error requirements, the TLP header is logged in the PCI Interface Error Log (PIE_LOGx) registers and the Intel XScale® microarchitecture are interrupted. When bit 5 is set in the ["PCI Express Advisory Error Control Register - PIE_AEC"](#), this is treated as an "Advisory Error" and an ERR_COR is sent to the root complex; otherwise an ERR_NONFATAL is issued.

For advisory errors, the firmware can reissue the transaction one or more (finite) times in an attempt to get valid data. When firmware decides to stop retrying the transaction it must escalate the error by setting the Generate ERR_NONFATAL bit in the ["PCI Express Advisory Error Control Register - PIE_AEC"](#).

Note: When the severity setting in ["PCI Express Uncorrectable Error Severity - ERRUNC_SEV"](#) register is fatal this is not an Advisory Error and an ERR_FATAL is sent to the root complex. Auto-recovery is discouraged as the ERR_FATAL message likely brings down the hierarchy.

3.9.1.8 Completion Timeout

When an out-bound non-posted request results in a completion timeout, the Advanced Error registers are updated and the initial request header and corresponding DMA descriptor tag are logged in the PCI Interface Error Log (PIE_LOGx) registers and the Intel XScale® microarchitecture are interrupted. When bit 6 is set in the ["PCI Express Advisory Error Control Register - PIE_AEC"](#), this is treated as an "Advisory Error" and an ERR_COR is sent to the root complex; otherwise an ERR_NONFATAL is issued.

For advisory errors, the firmware can reissue the transaction one or more (finite) times in an attempt to get valid data. When firmware decides to stop retrying the transaction it must escalate the error by setting the Generate ERR_NONFATAL bit in the ["PCI Express Advisory Error Control Register - PIE_AEC"](#).

Note: When the severity setting in ["PCI Express Uncorrectable Error Severity - ERRUNC_SEV"](#) register is fatal this is not an Advisory Error and an ERR_FATAL is sent to the root complex. Auto-recovery is discouraged as the ERR_FATAL message likely brings down the hierarchy.



3.9.2 Parity Error on the Internal Bus

The 81341 and 81342 provides support for byte-wise parity protection on the internal bus. The internal bus consists of a 36 bit address bus and 128 bit data bus; both are protected by byte-wise parity. The internal bus parity protection is provided independent of the operating mode of the ATU's PCI interface.

When initiating transactions on the internal bus, the ATU's internal bus interface generates byte-wise parity. As a target the ATU checks byte-wise parity.

3.9.3 ATU Error Summary

Table 131, "PCI Express Error Summary" summarizes the ATU error reporting for PCI Express Link errors, Table 132, "Root Complex Error Summary" summarizes the error reporting when operating as a Root Complex, and Table 133, "Internal Bus Error Summary" summarizes the ATU error reporting for internal bus errors. The tables assume that all error reporting is enabled through the appropriate command registers (unless otherwise noted).

Example: A poisoned TLP is received.

- Depending on the setting in the ERRUNC_SEV register, send ERR_FATAL/ NON_FATAL to the root complex.
- Log the error in the Advanced Error registers (ADVERR_CTL, ADVERR_LOG) as well as the PCI Interface Error registers (PIE_CSR, PIE_LOG, PIE_DLOG)
- Set the Detected Parity Error, SERR# Asserted, and possible the Master Data Parity Error bits in the ATUCR.
- Set the Fatal Error Detected or Non-Fatal Error Detected in the Device Status register (PE_DSTS) depending on the error message sent
- Set the Poisoned TLP Received status bit in the ERRUNC_STS register
- Set the Poisoned TLP Received status bit in the PIE_STST register.
- Depending on the mask bits, set any or all of the following bits in the ATUISR
 - PCI Interface Error Interrupt
 - Uncorrectable Error Transmitted Interrupt
 - Detected Parity Error Interrupt
 - Master Data Parity Error Interrupt
- Possible mask bits are the SERR# Enable (ATUCMD[8]), Fatal/Non-Fatal Error Reporting Enable (PE_DCTL[2 or 1]), Interrupt Masks (ATUIMR[8,4,0]) and logging masks (ERRUNC_MSK[12] & PIE_MSK[12]).



Table 131. PCI Express Error Summary (Sheet 1 of 2)

Error Condition	Bus Protocol Action ^a	Affected Logging Register	Affected bits in Unit Status Register	Affected bits in Interrupt Status Register	Unit Interrupt Mask Bits
Transaction Layer Errors					
Poisoned TLP Received	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[15, 14, 8] PE_DSTS[2 or 1] ERRUNC_STS[12] PIE_STS[12]	ATUISR[10, 8, 4, 0]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8, 4, 0] ERRUNC_MSK[12] PIE_MSK[12]
ECRC Check Failed	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[19] PIE_STS[19]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[19] PIE_MSK[19]
Unsupported Request	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[14] PE_DSTS[3] PE_DSTS[2 or 1] ERRUNC_STS[20] PIE_STS[20]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[20] PIE_MSK[20]
Completion Timeout	Requester: Send ERR_FATAL/ ERR_NONFATAL to Root Complex	PIE_CSR PIE_LOG[3:0] PIE_DLOG	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[14] PIE_STS[14]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[14] PIE_MSK[14]
Completer Abort	Completer: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[14, 11] PE_DSTS[2 or 1] ERRUNC_STS[15] PIE_STS[15]	ATUISR[10, 8, 5, 2]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8, 5, 2] ERRUNC_MSK[15] PIE_MSK[15]
Unexpected Completion	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[16] PIE_STS[16]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[16] PIE_MSK[16]
Receiver Overflow	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex	PIE_CSR PIE_LOG[3:0] PIE_DLOG	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[17] PIE_STS[17]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[17] PIE_MSK[17]
Flow Control Protocol Error	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex	PIE_CSR	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[13] PIE_STS[13]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[13] PIE_MSK[13]

**Table 131. PCI Express Error Summary (Sheet 2 of 2)**

Error Condition	Bus Protocol Action ^a	Affected Logging Register	Affected bits in Unit Status Register	Affected bits in Interrupt Status Register	Unit Interrupt Mask Bits
Malformed TLP	Receiver: Send ERR_FATAL/ ERR_NONFATAL to Root Complex Log the header of the TLP that caused the error.	ADVERR_CTL ADVERR_LOGx PIE_CSR PIE_LOGx PIE_DLOG	ATUSR[14] PE_DSTS[2 or 1] ERRUNC_STS[18] PIE_STS[18]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[18] PIE_MSK[18]
Physical Layer Errors					
Receiver Error	Send ERR_COR to Root Complex		PE_DSTS[0] ERRCOR_STS[0]	ATUISR[9]	PE_DCTL[0] ATUIMR[9]
Training Error	Send ERR_FATAL/ ERR_NONFATAL to Root Complex	PIE_CSR	PE_DSTS[2 or 1] ERRUNC_STS[0] PIE_STS[0]	ATUISR[8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8]
Data Link Layer Errors					
Bad TLP	Receiver: Send ERR_COR to Root Complex		PE_DSTS[0] ERRCOR_STS[6]	ATUISR[9]	PE_DCTL[0] ERRCOR_MSK[6] ATUIMR[9]
Bad DLLP	Receiver: Send ERR_COR to Root Complex		PE_DSTS[0] ERRCOR_STS[7]	ATUISR[9]	PE_DCTL[0] ERRCOR_MSK[7] ATUIMR[9]
Replay Timeout	Transmitter: Send ERR_COR to Root Complex		PE_DSTS[0] ERRCOR_STS[12]	ATUISR[9]	PE_DCTL[0] ERRCOR_MSK[12] ATUIMR[9]
REPLAY_NUM Rollover	Transmitter: Send ERR_COR to Root Complex		PE_DSTS[0] ERRCOR_STS[8]	ATUISR[9]	PE_DCTL[0] ERRCOR_MSK[8] ATUIMR[9]
Data Link Layer Protocol Error	Send ERR_FATAL/ ERR_NONFATAL to Root Complex	PIE_CSR	PE_DSTS[2 or 1] ERRUNC_STS[4] PIE_STS[4]	ATUISR[10, 8]	ATUCMD[8] PE_DCTL[2 or 1] ATUIMR[8] ERRUNC_MSK[4] PIE_MSK[4]
Device Specific Errors					
Received Completion with UR status	None	PIE_CSR PIE_LOG[3:0] PIE_DLOG	ATUSR[13] PIE_STS[31]	ATUISR[10, 3]	ATUIMR[3] PIE_MSK[31]
Received Completion with CA status	None	PIE_CSR PIE_LOG[3:0] PIE_DLOG	ATUSR[12] PIE_STS[30]	ATUISR[10, 1]	ATUIMR[1] PIE_MSK[30]
Poisoned TLP Transmitted	None	PIE_CSR PIE_LOG[3:0] PIE_DLOG	ATUSR[8] PIE_STS[29]	ATUISR[10, 0]	ATUIMR[0] PIE_MSK[29]
Outbound Header Parity Error detected	None	PIE_CSR PIE_LOG[3:0] PIE_DLOG	PIE_STS[28]	ATUISR[10]	PIE_MSK[28]

a. ERR_FATAL / ERR_NONFATAL action is determined by the severity bits in the ERRUNC_SEV register.

**Table 132. Root Complex Error Summary**

Error Condition	Bus Protocol Action	Affected Logging Register	Affected bits in Unit Status Register	Affected bits in Interrupt Status Register	Unit Interrupt Mask Bits
Root Complex Errors					
Received ERR_COR	None	RERR_ID	RERR_SR[1, 0]	ATUISR[12, 11]	ATUIMR[12] RERR_CMD[0]
Received ERR_NONFATAL	None	RERR_ID	RERR_SR[5, 4, 3, 2]	ATUISR[12, 11]	ATUIMR[12] RERR_CMD[1]
Received ERR_FATAL	None	RERR_ID	RERR_SR[6, 4, 3, 2]	ATUISR[12, 11]	ATUIMR[12] RERR_CMD[2]

Table 133. Internal Bus Error Summary

Error Condition	Bus Protocol Action	Affected Logging Register	Affected bits in Unit Status Register	Affected bits in Interrupt Status Register	Unit Interrupt Mask Bits
Internal Bus Errors					
Master Abort on inbound requests	Signal Completer Abort	See Completer Abort above	See Completer Abort above	<i>Completer Abort and</i> ATUISR[5]	<i>Completer Abort and</i> ATUIMR[5]
AERR on inbound requests	Signal Completer Abort	See Completer Abort above	See Completer Abort above	<i>Completer Abort and</i> ATUISR[5]	<i>Completer Abort and</i> ATUIMR[5]
Target Abort on inbound requests	Signal Completer Abort	See Completer Abort above	See Completer Abort above	See Completer Abort above	See Completer Abort above
Data Parity Error on outbound Writes or Completions	Set EP bit in TLP Header. Posted Writes may get flushed when bit 8 of ATUCR is set.	See Poisoned TLP Transmitted above	See Poisoned TLP Transmitted above	See Poisoned TLP Transmitted above	See Poisoned TLP Transmitted above
Header Parity Error in Link Layer Retry Buffer	TLP transmitted as Malformed. Posted Writes may get flushed when bit 8 of ATUCR is set.	See Outbound Header Parity Error above			INTPND3[3]



3.10 PCI Express* Hot-Plug Support

The PCI Express architecture is designed to natively support both Hot-Plug and hot remove of devices. This section defines the usage model defined for all the ATU.

ATU supports the receipt and generation of Hot-Plug messages via the Inbound/Outbound message header registers. When a Hot-Plug message is received, the core can then utilize the GPIOs to change the indicator lights, or conversely sample the switch status and generate a message as appropriate.



3.11 Reset

The PCI Express specification defines three distinct types of reset: cold, warm, and hot.

The fundamental reset that occurs following initial power on is considered a hot reset. The assertion of the PRST# or WARM_RST# pins are considered warm resets. The receipt of the inband reset training sequence is considered a hot reset.

Sticky bits are preserved under the WARM_RST# and inband hot reset conditions.

Additional reset details are specified in [Chapter 19.0, "Clocking and Reset"](#).



3.12 Message-Signaled Interrupts

The Messaging Unit is responsible for the generation of all of the Outbound Interrupts from the 81341 and 81342. These interrupts can be delivered to the Host Processor via the legacy Assert_INTx/Deassert_INTx messages or the Message Signaled Interrupt (MSI) mechanism.

When a host processor enables Message-Signaled Interrupts (MSI) on the 81341 and 81342, an outbound interrupt is signaled to the host via a posted write instead of the legacy Assert/Deassert messages.

In support of MSI, the 81341 and 81342 implements the MSI capability structure. The capability structure includes the [Section 4.9.30, "MSI Capability Identifier Register - Cap_ID" on page 450](#), the [Section 4.9.31, "MSI Next Item Pointer Register - MSI_Next_Ptr" on page 451](#), the [Section 4.9.33, "Message Address Register - Message_Address" on page 453](#), the [Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454](#) and the [Section 4.9.35, "Message Data Register- Message_Data" on page 455](#).

The Message Unit generates MSIs by writing to the MSI port via the internal bus. The ATU generates a write transaction whenever the Message Unit writes to the MSI port, using the address specified in the [Section 4.9.33, "Message Address Register - Message_Address" on page 453](#) and [Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454](#) and the data provided in the [Section 4.9.35, "Message Data Register- Message_Data" on page 455](#).

3.12.1 Legacy Interrupts

The ATU supports the generation of the legacy Assert_INTx/Deassert_INTx interrupt messages.

3.12.2 Internal Interrupts

The ATU has 4 internal interrupts that connect to the internal Interrupt Controller Unit.

ATU Error Interrupt

ATU Inbound Message Interrupt

ATU Configuration Write Interrupt

ATU BIST Interrupt



3.13 Vital Product Data

Vital Product Data (VPD) provides detailed information to the system regarding the hardware, software and microcode elements of a device. This information may include Part Number, Serial Number or other detailed information. This information resides on a non-volatile storage device (i.e., Flash Memory) attached to the 81341 and 81342. In addition VPD also provides a mechanism for storing information such as performance or failure data on the device being monitored.

Support of VPD involves the implementation of the VPD Extended Capabilities List Item in the Primary ATU. The VPD Extended capabilities header consists of four registers, the ["VPD Capability Identifier Register - VPD_Cap_ID" on page 339](#), the ["VPD Next Item Pointer Register - VPD_Next_Item_Ptr" on page 339](#), the ["VPD Address Register - VPDAR" on page 340](#), and the ["VPD Data Register - VPDDR" on page 340](#).

Scheduled by Intel XScale® processor interrupts, the 81341 and 81342 may be used to retrieve or store VPD information through the VPD extended capabilities list item.

Please consult Appendix I of the *PCI Local Bus Specification*, Revision 2.3 for the definitions of compliant VPD format.

3.13.1 Configuring Vital Product Data Operation

By default, the 81341 and 81342 VPD functionality is not configured for operation. Specifically, the VPD Extended Capabilities List Item are not discovered during a PCI Express Link scan and the ATUs VPD interrupt status bit in the ["ATU Interrupt Status Register - ATUISR" on page 333](#) are masked by the ["ATU Interrupt Mask Register - ATUIMR" on page 336](#). The following steps should be followed to properly configure the 81341 and 81342's support for VPD:

1. The 81341 and 81342 must be strapped to Retry Type 0 Configuration cycles following the deassertion of **P_RST#**. Enabling this configuration cycle retry mechanism insures that the Intel XScale® processor can make the VPD Extended Capabilities List Item visible before the system configures the 81341 and 81342. The configuration retry mechanism is controlled through bit 2 of the ["PCI Configuration and Status Register - PCSR" on page 331](#).
2. When the configuration retry mechanism is strapped enabled as described in step 1, typically, the 81341 and 81342 would also be strapped such that the Intel XScale® processor would immediately boot following the deassertion of **P_RST#** (bit 1 of the PCSR), though this is not required.
3. The Intel XScale® processor writes 90H to the ["PCI Express Next Item Pointer Register - PCIE_NXTP" on page 345](#). This links the PCI Express Capabilities List Item to the VPD Capabilities List Item.
4. The Intel XScale® processor clears bit 17 of the ATUIMR to enable the ATUs VPD interrupt status bit.



3.13.2 Accessing Vital Product Data

The VPD Capabilities List Item provides three fields which the system uses to access the Vital Product Data:

VPD Address	DWORD Aligned Byte address of the VPD to be accessed which is represented by VPDAR[14:0]. Note that this means that the maximum size of the VPD is 128 Kbytes. The user may pick any 128 Kbyte block of memory in the storage component for the VPD.
Flag	The flag register is used to indicate when the transfer between the VPD Data Register and the storage component is completed. The flag is in VPDAR[15] which means that the Flag is written at the same time that VPD address is written.
VPD Data	Four bytes of VPD Data can be read or written through this field which is represented by VPDDR[31:0]. The least significant byte of this register represents the byte at the VPD Address (VPDAR[14:0]). Four bytes are always transferred between this register and the VPD storage component.

3.13.2.1 Reading Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 81341 and 81342 reads Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag cleared.
2. An interrupt to the Intel XScale® processor is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be set.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being set, the results of the original read operation are unpredictable.

3. Using the VPD Address, the Intel XScale® processor retrieves the Vital Product Data from the VPD storage component (i.e., Flash Memory).
4. The Intel XScale® processor then writes this data to VPD Data Register (VPDDR).
5. The Intel XScale® processor clears the VPD interrupt status bit in the ATUISR.
6. The Intel XScale® processor then sets the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been set, the host processor then reads the retrieved VPD from the VPDDR.



3.13.2.2 Writing Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 81341 and 81342 writes Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD data to be written to the VPDDR.
2. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag set.
3. An interrupt to the Intel XScale® processor is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be cleared.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being cleared, the results of the original write operation are unpredictable.

4. Using the VPD Address, the Intel XScale® processor writes the Vital Product Data from the VPDDR to the VPD storage component (i.e., Flash Memory).
5. The Intel XScale® processor clears the VPD interrupt status bit in the ATUISR.
6. The Intel XScale® processor then clears the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been cleared, the host processor has been informed that the VPD write operation is complete.



3.14 Root Complex Functionality

The ATU has the capability to operate as the Root Complex of the PCI Express hierarchy.

When operating as a Root Complex, the ATU behaves as a single function device and only claims transactions for the ATU function. This means that the BARs and control signals from other internal functions are ignored and any PCI status updates are limited to the ATU.

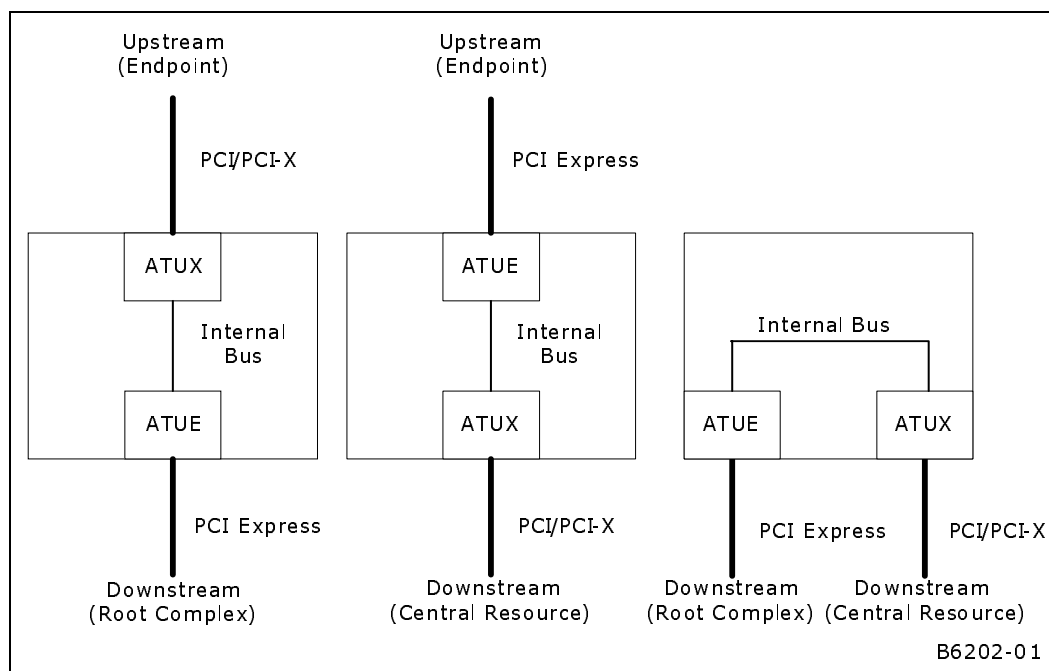
Note: When an endpoint device transmits an upstream configuration cycle to the ATUE as a Root Complex, the ATUE will discard the configuration cycle without sending a completion.

3.15 Embedded Bridge Functionality

The dual interface 81341 and 81342 provides two Address Translation Units: the ATUE provides a PCI Express interface, and the ATUX provides a PCI/PCIX interface. The two ATUs are internally connected via the south internal bus.

The two ATUs can be setup and used as an embedded bridge (non-transparent bridge). For example, the dual interface 81341 and 81342 can be used to bridge a PCI Express device and a PCI-X device in a system. Both ATUs support endpoint mode and root complex or central resource mode capabilities. Therefore, either ATU can be connected upstream or downstream in a system. [Figure 33](#) shows the possible configurations of the dual interface 81341 and 81342.

Figure 33. Intel® 81341 and 81342 I/O Processors PCI Interface Configuration Examples



To allow PCI transactions to flow between the two PCI interfaces of the dual interface 81341 and 81342 in a bidirectional manner, each of the ATUs must have at least one of its Inbound Address Translation Windows mapped to one of the other ATU's Outbound Address Translation Windows. Refer to [Section 3.3.1, "Inbound Transactions" on page 242](#) and [Section 3.3.2, "Outbound Transactions" on page 249](#) that explain how the Inbound and Outbound address translation windows operate. Below are few examples that demonstrate how the ATUs may be programmed to operate as an embedded bridge.



Example 6. ATUE as an Endpoint and ATUX as a Central Resource

The user wants to setup the ATUE as an endpoint and ATUX as a central resource by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 9 and the ATUX outbound window is placed in section 8 of the 64-GByte internal address space.

- External Straps: INTERFACE_SEL_PCIX# = '1', PCIE_RC# = '1', and PCIX_EP# = '1'. Note that setting up INTERFACE_SEL_PCIX# to '1' causes the ATUE to be function number 0H and the ATUX to be function number 5H. The function number must be consistent with the function number field programmed in the OUMBAR2[30:28] as this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: IALR2 = 00000000H, IATVR2 = 00000000H, IAUTVR2 = 00000008H.
- Outbound Address Translation Window for ATUX: OUMBAR2 = D0000008H, OUMWTVR2 = 00000000H
- Inbound Address Translation Window for ATUX: IALR2 = 00000000H, IATVR2 = 00000000H, IAUTVR2 = 00000009H.
- Outbound Address Translation Window for ATUE: OUMBAR2 = 80000009H, OUMWTVR2 = 00000000H.

Example 7. ATUE as a Root Complex and ATUX as an Endpoint

The user wants to setup the ATUE as root complex and ATUX as an endpoint by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 7 and the ATUX outbound window is placed in section 6 of the 64-GByte internal address space.

- External Straps: INTERFACE_SEL_PCIX# = '0', PCIE_RC# = '0', and PCIX_EP# = '0'. Note that setting up INTERFACE_SEL_PCIX# to '0' causes the ATUX to be function number 0H and the ATUE to be function number 5H. The function number must be consistent with the function number field programmed in the OUMBAR2[30:28] as this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: IALR2 = 00000000H, IATVR2 = 00000000H, IAUTVR2 = 00000006H.
- Outbound Address Translation Window for ATUX: OUMBAR2 = 80000006H, OUMWTVR2 = 00000000H
- Inbound Address Translation Window for ATUX: IALR2 = 00000000H, IATVR2 = 00000000H, IAUTVR2 = 00000007H.
- Outbound Address Translation Window for ATUE: OUMBAR2 = D0000007H, OUMWTVR2 = 00000000H.

**Example 8. ATUE as a Root Complex and ATUX as a Central Resource**

The user wants to setup the ATUE as root complex and ATUX as a central resource by using the Inbound Address Translation Window 2 and Outbound Address Translation Window 2 of the ATUs. Each ATU would support a 4-GByte Inbound window and a 4-GByte Outbound window. ATUE outbound window is placed in section 6 and the ATUX outbound window is placed in section 9 of the 64-GByte internal address space.

- External Straps: `INTERFACE_SEL_PCIX#` = '0', `PCIE_RC#` = '0', and `PCIX_EP#` = '1'. Note that setting up `INTERFACE_SEL_PCIX#` to '0' causes the ATUX to be function number 0H and the ATUE to be function number 5H. In this scenario since none of the ATUs is an endpoint, `INTERFACE_SEL_PCIX#` can be setup to either '0' or '1'. However, the function number must be consistent with the function number field programmed in the `OUMBAR2[30:28]` as this function number is used to report and log error conditions.
- Inbound Address Translation Window for ATUE: `IALR2` = 00000000H, `IATVR2` = 00000000H, `IAUTVR2` = 00000009H.
- Outbound Address Translation Window for ATUX: `OUMBAR2` = 80000009H, `OUMWTVR2` = 00000000H
- Inbound Address Translation Window for ATUX: `IALR2` = 00000000H, `IATVR2` = 00000000H, `IAUTVR2` = 00000006H.
- Outbound Address Translation Window for ATUE: `OUMBAR2` = D0000006H, `OUMWTVR2` = 00000000H.

Warning: The following restrictions on the embedded bridge transactions from the ATUX to ATUE need to be noted.

- When the ATUX is operating in PCI-X mode, Inbound Write transactions with non-contiguous byte enables are not supported. Note that byte enables may be disabled in the first and/or last DWORD of an Inbound Write transaction.
- When the ATUX is operating in conventional PCI mode, all of the byte enables must be asserted on Inbound Write transactions.



3.16 Register Definitions

Every PCI device implements its own separate configuration address space and configuration registers. The PCI Express Specification extends the configuration space to 4096 bytes as compared to 256 bytes allowed by *PCI Local Bus Specification*, Revision 2.3. The ATU configuration space is divided into a PCI 2.3 compatible region consisting of the first 256 bytes and an extended PCI express configuration space region consisting of the remaining space. The PCI 2.3 compatible space provides support for legacy operating systems. The first 64 bytes must adhere to a predefined header format.

Figure 34 defines the header format. Table 135 shows the PCI configuration registers, listed by internal bus address offset. Table 135 shows the entire ATU configuration space (including header and extended registers) and the corresponding section that describes each register. Note that all configuration read and write transactions is accepted on the internal bus as 32-bit transactions. Refer to Chapter 21.0, “Peripheral Registers”.

Figure 34. ATU Interface Configuration Header Format

ATU Device ID		Vendor ID		00H
Status		Command		04H
ATU Class Code			Revision ID	08H
ATUBISTR	Header Type	Latency Timer	Cacheline Size	0CH
Inbound ATU Base Address 0				10H
Inbound ATU Upper Base Address 0				14H
Inbound ATU Base Address 1				18H
Inbound ATU Upper Base Address 1				1CH
Inbound ATU Base Address 2				20H
Inbound ATU Upper Base Address 2				24H
Reserved				28H
ATU Subsystem ID		ATU Subsystem Vendor ID		2CH
Expansion ROM Base Address				30H
Reserved			Capabilities Pointer	34H
Reserved				38H
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	3CH

The ATU is programmed via a Type 0 configuration command on the PCI interface. See [Section 3.3.1.5, “Inbound Configuration Cycle Translation \(ID Routed\)” on page 247](#). ATU configuration space is function number zero of the 81341 and 81342 single-function PCI device.

Beyond the required 64 byte header format, ATU configuration space implements extended register space in support of the units functionality. Refer to the *PCI Express Base Specification*, Revision 1.0a for details on accessing and programming configuration register space.

3.16.1 Extended Capabilities Registers

The ATU unit includes 5 extended capability configuration spaces beginning at configuration offset 90H, 98H, A0H, B0H, and D0H. The extended configuration spaces can be accessed by a device on the PCI interface through a mechanism defined in the PCI Express Specification.

In the ATU Status Register (Section 3.16.6) the appropriate bit is set indicating that the Extended Capability Configuration space is supported. When this bit is read, the device can then read the Capabilities Pointer register (Section 3.16.23) to determine the configuration offset of the Extended Capabilities Configuration Header. The format of these headers are depicted in Figure 35, Figure 37, Figure 38 and Figure 39.

Figure 35. ATU Interface Extended Configuration Header Format (Power Management)

Power Management Capabilities	Next Item Pointer	Capability Identifier	98H
Reserved	Power Management Control/Status		9CH

The first byte at the Extended Configuration Offset 98H is the ATU Capability Identifier Register for the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 3.16.51, “PM Next Item Pointer Register - PM_Next_Item_Ptr” on page 341) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to D0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

To enable the *PCI Bus Power Management Interface Specification*, Revision 1.1 compliance support, the Power State Transition interrupt mask in bit 8 of the ATUIMR needs to be cleared. It is the configuration software’s responsibility to properly enable and initialize the ATUs Power Management Interface before the Configuration Cycle Retry Bit in the Section 3.16.41, “PCI Configuration and Status Register - PCSR” on page 331 is cleared in order for the ATU to be *Advanced Configuration and Power Interface Specification*, Revision 2.0 compliant.

Figure 36. ATU Interface Extended Configuration Header Format (MSI-X Capability)

MSI-X Message Control	MSI-X Next Item Pointer	MSI-X Capability ID	B0H
MSI-X Table Offset		Table BIR	B4H
MSI-X PBA Offset		PBA BIR	B8H

Note: MSI-X Capability Registers are defined in Chapter 4.0, “Messaging Unit.”

The first byte at the Extended Configuration Offset B0H is the MSI-X Capability Identifier Register (Section 4.9.36, “MSI-X Capability Identifier Register - MSI-X_Cap_ID”). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 4.9.37, “MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr”) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to A0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

**Figure 37. ATU Interface Extended Configuration Header Format (MSI Capability)**

MSI Message Control	MSI Next Item Pointer	MSI Capability ID	A0H
MSI Message Address			A4H
MSI Message Upper Address			A8H
Reserved	MSI Message Data		ACH

The first byte at the Extended Configuration Offset D0H is the [Section 4.9.30, “MSI Capability Identifier Register - Cap_ID” on page 450](#). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Next Item Pointer Register which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to D0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 38. ATU Interface Extended Configuration Header Format (PCI Express Capability)

PCI Express Capabilities Register	Next Item Pointer	PCI Express Cap ID	D0H
PCI Express Device Capabilities			D4H
PCI Express Device Status	PCI Express Device Control		D8H
PCI Express Link Capabilities			DCH
PCI Express Link Status	PCI Express Link Control		E0H
PCI Express Slot Capabilities			E4H
PCI Express Slot Status	PCI Express Slot Control		E8H
Reserved	PCI Express Root Control		ECH
PCI Express Root Status			

The first byte at the Extended Configuration Offset D0H is the PCI Express Capability Identifier Register ([Section 3.16.55](#)). This identifies this Extended Configuration Header space as the type defined by the *PCI Express Base Specification*, Revision 1.0a.

Following the Capability Identifier Register is the single byte Next Item Pointer Register ([Section 3.16.56](#)) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H by default to indicate there are no additional Extended Capabilities Headers in the ATUs configuration space. Software can set this pointer to 90H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 39. ATU Interface Extended Configuration Header Format (VPD Capability)

VPD Address	Next Item Pointer	VPD Capability ID	90H
VPD Data			94H



The first byte at the Extended Configuration Offset 90H is the VPD Capability Identifier Register (Section 3.16.44). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 3.16.47) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H indicating that there are no additional Extended Capabilities Headers supported in the ATUs configuration space.

The following sections describe the ATU and Expansion ROM configuration registers. Configuration space consists of 8, 16, 24, and 32-bit registers arranged in a predefined format. Each register is described in functionality, access type (read/write, read/clear, read only) and reset default condition.

See Section 1.6, “Terminology and Conventions” on page 53 for a description of *reserved*, *read only*, and *read/clear*. All registers adhere to the definitions found in the *PCI Local Bus Specification*, Revision 2.3 unless otherwise noted.

The PCI register number for each register is given in Table 135.

Note:

Each configuration register’s access type is individually defined for PCI configuration accesses. Some PCI read-only configuration registers have read/write capability from the 81341 and 81342 core CPU. See also Chapter 21.0, “Peripheral Registers”.



3.16.2 Internal Bus Addresses

All of the ATU registers are accessible through both inbound PCI configuration cycles and the 81341 and 81342 core CPU (Register offsets 000H through 0FFH). T.

The location of these registers are specified as a relative offset to a 512KB aligned global PMMR offset. The default for the 512KB aligned offset is 0 FFD8 0000H defined by the PMMRBAR register. See also [Chapter 21.0, “Peripheral Registers”](#).

The Internal Bus Address Offset to PMMRBAR of any ATU Register can be derived by adding the 4 KB address aligned Internal Bus Memory Mapped Register Range Offset ([Table 134, “ATU Internal Bus Memory Mapped Register Range Offsets” on page 297](#)) to the Register Offset ([Table 135, “ATU PCI Configuration Register Space” on page 297](#))

For example, when INTERFACE_SEL_PCIX# is asserted, the offset to PMMRBAR of the “ATU Command Register - ATUCMD” would be (4 D000H+004H) or 4 D004H.

Note: The 4 KB Address Aligned Range Offset can be different depending on two configuration straps as described in [Table 134](#).

Table 134. ATU Internal Bus Memory Mapped Register Range Offsets

INTERFACE_SEL_PCIX#	Internal Bus MMR Address Range Offset (Relative to PMMRBAR)
Deasserted (1)	+4 8000H
Asserted (0)	+4 D000H

Table 135. ATU PCI Configuration Register Space (Sheet 1 of 4)

Internal Bus Address Offset	ATU PCI Configuration Register Section, Name, Page
+000H	Section 3.16.3, “ATU Vendor ID Register - ATUVID” on page 301
+002H	Section 3.16.4, “ATU Device ID Register - ATUDID” on page 301
+004H	Section 3.16.5, “ATU Command Register - ATUCMD” on page 302
+006H	Section 3.16.6, “ATU Status Register - ATUSR” on page 303
+008H	Section 3.16.7, “ATU Revision ID Register - ATURID” on page 304
+009H	Section 3.16.8, “ATU Class Code Register - ATUCCR” on page 304
+00CH	Section 3.16.9, “ATU Cacheline Size Register - ATUCLSR” on page 305
+00DH	Section 3.16.10, “ATU Latency Timer Register - ATULT” on page 305
+00EH	Section 3.16.11, “ATU Header Type Register - ATUHTR” on page 306
+00FH	Section 3.16.12, “ATU BIST Register - ATUBISTR” on page 307
+010H	Section 3.16.13, “Inbound ATU Base Address Register 0 - IABAR0” on page 308
+014H	Section 3.16.14, “Inbound ATU Upper Base Address Register 0 - IAUBAR0” on page 309
+018H	Section 3.16.16, “Inbound ATU Base Address Register 1 - IABAR1” on page 312
+01CH	Section 3.16.17, “Inbound ATU Upper Base Address Register 1 - IAUBAR1” on page 313
+020H	Section 3.16.18, “Inbound ATU Base Address Register 2 - IABAR2” on page 314
+024H	Section 3.16.19, “Inbound ATU Upper Base Address Register 2 - IAUBAR2” on page 315
+02CH	Section 3.16.20, “ATU Subsystem Vendor ID Register - ASVIR” on page 316
+02EH	Section 3.16.21, “ATU Subsystem ID Register - ASIR” on page 316
+030H	Section 3.16.22, “Expansion ROM Base Address Register - ERBAR” on page 317
+034H	Section 3.16.23, “ATU Capabilities Pointer Register - ATU_Cap_Ptr” on page 318



Table 135. ATU PCI Configuration Register Space (Sheet 2 of 4)

Internal Bus Address Offset	ATU PCI Configuration Register Section, Name, Page
+03CH	Section 3.16.24, "ATU Interrupt Line Register - ATUILR" on page 319
+03DH	Section 3.16.25, "ATU Interrupt Pin Register - ATUIPR" on page 320
+03EH	Section 3.16.26, "ATU Minimum Grant Register - ATUMGNT" on page 320
+03FH	Section 3.16.27, "ATU Maximum Latency Register - ATUMLAT" on page 321
+040H	Section 3.16.28, "Inbound ATU Limit Register 0 - IALR0" on page 322
+044H	Section 3.16.29, "Inbound ATU Translate Value Register 0 - IATVR0" on page 323
+048H	Section 3.16.30, "Inbound ATU Upper Translate Value Register 0 - IAUTVR0" on page 323
+04CH	Section 3.16.31, "Inbound ATU Limit Register 1 - IALR1" on page 324
+050H	Section 3.16.32, "Inbound ATU Translate Value Register 1 - IATVR1" on page 325
+054H	Section 3.16.33, "Inbound ATU Upper Translate Value Register 1 - IAUTVR1" on page 325
+058H	Section 3.16.34, "Inbound ATU Limit Register 2 - IALR2" on page 326
+05CH	Section 3.16.35, "Inbound ATU Translate Value Register 2 - IATVR2" on page 327
+060H	Section 3.16.36, "Inbound ATU Upper Translate Value Register 2 - IAUTVR2" on page 328
+064H	Section 3.16.37, "Expansion ROM Limit Register - ERLR" on page 328
+068H	Section 3.16.38, "Expansion ROM Translate Value Register - ERTVR" on page 329
+06CH	Section 3.16.39, "Expansion ROM Upper Translate Value Register - ERUTVR" on page 329
+070H	Section 3.16.40, "ATU Configuration Register - ATUCR" on page 330
+074H	Section 3.16.41, "PCI Configuration and Status Register - PCSR" on page 331
+078H	Section 3.16.42, "ATU Interrupt Status Register - ATUISR" on page 333
+07CH	Section 3.16.43, "ATU Interrupt Mask Register - ATUIMR" on page 336
+080H	Section 3.16.44, "PCI Express Message Control/Status Register - PEMCSR" on page 337
+084H	Section 3.16.45, "PCI Express Link Control/Status Register - PELCSR" on page 338
+090H	Section 3.16.46, "VPD Capability Identifier Register - VPD_Cap_ID" on page 339
+091H	Section 3.16.47, "VPD Next Item Pointer Register - VPD_Next_Item_Ptr" on page 339
+092H	Section 3.16.48, "VPD Address Register - VPDAR" on page 340
+094H	Section 3.16.49, "VPD Data Register - VPDDR" on page 340
+098H	Section 3.16.50, "PM Capability Identifier Register - PM_Cap_ID" on page 341
+099H	Section 3.16.51, "PM Next Item Pointer Register - PM_Next_Item_Ptr" on page 341
+09AH	Section 3.16.52, "ATU Power Management Capabilities Register - APMCR" on page 342
+09CH	Section 3.16.53, "ATU Power Management Control/Status Register - APMCSR" on page 343
+0A0H	Section 4.9.30, "MSI Capability Identifier Register - Cap_ID" on page 450 ^a
+0A1H	Section 4.9.31, "MSI Next Item Pointer Register - MSI_Next_Ptr" on page 451 ^a
+0A2H	Section 4.9.32, "Message Control Register - Message_Control" on page 452 ^a
+0A4H	Section 4.9.33, "Message Address Register - Message_Address" on page 453 ^a
+0A8H	Section 4.9.34, "Message Upper Address Register - Message_Upper_Address" on page 454 ^a
+0ACH	Section 4.9.35, "Message Data Register - Message_Data" on page 455 ^a
+0B0H	Section 4.9.36, "MSI-X Capability Identifier Register - MSI-X_Cap_ID" on page 456 ^b
+0B1H	Section 4.9.37, "MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr" on page 457 ^b
+0B2H	Section 4.9.38, "MSI-X Message Control Register - MSI-X_MCR" on page 458 ^b
+0B4H	Section 4.9.39, "MSI-X Table Offset Register — MSI-X_Table_Offset" on page 459 ^b
+0B8H	Section 4.9.40, "MSI-X Pending Bit Array Offset Register - MSI-X_PBA_Offset" on page 460 ^b
+0BCJ- +0C8H	Reserved
+0CCH	Section 3.16.54, "ATU Scratch Pad Register - ATUSPR" on page 344
+0D0H	Section 3.16.55, "PCI Express Capability List Register - PCIE_CAPID" on page 344

**Table 135. ATU PCI Configuration Register Space (Sheet 3 of 4)**

Internal Bus Address Offset	ATU PCI Configuration Register Section, Name, Page
+0D1H	Section 3.16.56, "PCI Express Next Item Pointer Register - PCIE_NXTP" on page 345
+0D2H	Section 3.16.57, "PCI Express Capabilities Register - PCIE_CAP" on page 346
+0D4H	Section 3.16.58, "PCI Express Device Capabilities Register - PCIE_DCAP" on page 347
+0D8H	Section 3.16.59, "PCI Express Device Control Register - PE_DCTL" on page 348
+0DAH	Section 3.16.60, "PCI Express Device Status Register - PE_DSTS" on page 350
+0DCH	Section 3.16.61, "PCI Express Link Capabilities Register - PE_LCAP" on page 351
+0E0H	Section 3.16.62, "PCI Express Link Control Register - PE_LCTL" on page 352
+0E2H	Section 3.16.63, "PCI Express Link Status Register - PE_LSTS" on page 353
+0E4H	Section 3.16.64, "PCI Express Slot Capabilities Register - PE_SCAP" on page 354
+0E8H	Section 3.16.65, "PCI Express Slot Control Register - PE_SCR" on page 355
+0EAH	Section 3.16.66, "PCI Express Slot Status Register - PE_SSTS" on page 356
+0ECH	Section 3.16.67, "PCI Express Root Control Register - PE_RCR" on page 357
+0F0H	Section 3.16.68, "PCI Express Root Status Register - PE_RSR" on page 358
+100H	Section 3.16.69, "PCI Express Advanced Error Capability Identifier - ADVERR_CAPID" on page 359
+104H	Section 3.16.70, "PCI Express Uncorrectable Error Status - ERRUNC_STS" on page 360
+108H	Section 3.16.71, "PCI Express Uncorrectable Error Mask - ERRUNC_MSK" on page 361
+10CH	Section 3.16.72, "PCI Express Uncorrectable Error Severity - ERRUNC_SEV" on page 362
+110H	Section 3.16.73, "PCI Express Correctable Error Status - ERRCOR_STS" on page 363
+114H	Section 3.16.74, "PCI Express Correctable Error Mask - ERRCOR_MSK" on page 364
+118H	Section 3.16.75, "Advanced Error Control and Capability Register - ADVERR_CTL" on page 365
+11CH	Section 3.16.76, "PCI Express Advanced Error Header Log - ADVERR_LOG0" on page 365
+120H	Section 3.16.77, "PCI Express Advanced Error Header Log - ADVERR_LOG1" on page 366
+124H	Section 3.16.78, "PCI Express Advanced Error Header Log - ADVERR_LOG2" on page 366
+128H	Section 3.16.79, "PCI Express Advanced Error Header Log - ADVERR_LOG3" on page 367
+12CH	Section 3.16.80, "Root Error Command Register - RERR_CMD" on page 367
+130H	Section 3.16.81, "Root Error Status Register" on page 368
+134H	Section 3.16.82, "Error Source Identification Register - RERR_ID" on page 369
+1E0H	Section 3.16.83, "Device Serial Number Capability - DSN_CAP" on page 369
+1E4H	Section 3.16.84, "Device Serial Number Lower DW Register - DSN_LDW" on page 370
+1E8H	Section 3.16.85, "Device Serial Number Upper DW Register - DSN_UDW" on page 370
+1ECH	Section 3.16.86, "PCI Express Advisory Error Control Register - PIE_AEC" on page 371
+1F0H	Section 3.16.87, "Power Budgeting Enhanced Capability Header - PWRBGT_CAPID" on page 372
+1F4H	Section 3.16.88, "Power Budgeting Data Select Register - PWRBGT_DSEL" on page 372
+1F8H	Section 3.16.89, "Power Budgeting Data Register - PWRBGT_DATA" on page 373
+1FCH	Section 3.16.90, "Power Budgeting Capability Register - PWRBGT_CAP" on page 374
+200H - +25FH	Section 3.16.91, "Power Budgeting Information Registers[0:23]—PWRBGT_INFO[0:23]" on page 375
+300H	Section 3.16.92, "Outbound I/O Base Address Register - OIOBAR" on page 376
+304H	Section 3.16.93, "Outbound I/O Window Translate Value Register - OIOWTVR" on page 377
+308H	Section 3.16.94, "Outbound Upper Memory Window Base Address Register 0 - OUMBAR0" on page 378
+30CH	Section 3.16.95, "Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0" on page 379
+310H	Section 3.16.96, "Outbound Upper Memory Window Base Address Register 1 - OUMBAR1" on page 380
+314H	Section 3.16.97, "Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1" on page 381
+318H	Section 3.16.98, "Outbound Upper Memory Window Base Address Register 2 - OUMBAR2" on page 382
+31CH	Section 3.16.99, "Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2" on page 383

**Table 135. ATU PCI Configuration Register Space (Sheet 4 of 4)**

Internal Bus Address Offset	ATU PCI Configuration Register Section, Name, Page
+320H	Section 3.16.100, "Outbound Upper Memory Window Base Address Register 3 - OUMBAR3" on page 384
+324H	Section 3.16.101, "Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3" on page 385
+328H	Reserved
+32CH	Section 3.16.102, "Outbound Configuration Cycle Address Register - OCCAR" on page 386
+330H	Section 3.16.103, "Outbound Configuration Cycle Data Register - OCCDR" on page 387
+334H	Section 3.16.104, "Outbound Configuration Cycle Function Number - OCCFN" on page 388
+340H	Section 3.16.105, "Inbound Vendor Message Header Register 0 - IVMHR0" on page 389
+344H	Section 3.16.106, "Inbound Vendor Message Header Register 1 - IVMHR1" on page 390
+348H	Section 3.16.107, "Inbound Vendor Message Header Register 2 - IVMHR2" on page 391
+34CH	Section 3.16.108, "Inbound Vendor Message Header Register 3 - IVMHR3" on page 392
+350H	Section 3.16.109, "Inbound Vendor Message Payload Register - IVMPR" on page 392
+360H	Section 3.16.110, "Outbound Vendor Message Header Register 0 - OVMHR0" on page 393
+364H	Section 3.16.111, "Outbound Vendor Message Header Register 1 - OVMHR1" on page 394
+368H	Section 3.16.112, "Outbound Vendor Message Header Register 2 - OVMHR2" on page 395
+36CH	Section 3.16.113, "Outbound Vendor Message Header Register 3 - OVMHR3" on page 395
+370H	Section 3.16.114, "Outbound Vendor Message Payload Register - OVMPR" on page 396
+380H	Section 3.16.115, "PCI Interface Error Control and Status Register - PIE_CSR" on page 397
+384H	Section 3.16.116, "PCI Interface Error Status - PIE_STS" on page 398
+388H	Section 3.16.117, "PCI Interface Error Mask - PIE_MSK" on page 399
+38CH	Section 3.16.118, "PCI Interface Error Header Log - PIE_LOG0" on page 400
+390H	Section 3.16.119, "PCI Interface Error Header Log 1 - PIE_LOG1" on page 400
+394H	Section 3.16.120, "PCI Interface Error Header Log 2 - PIE_LOG2" on page 401
+398H	Section 3.16.121, "PCI Interface Error Header Log - PIE_LOG3" on page 401
+39CH	Section 3.16.122, "PCI Interface Error Descriptor Log" on page 402
+3B0H	Section 3.16.123, "ATU Reset Control Register - ATURCR" on page 402

- a. Refer to the Messaging Unit Chapter for MSI Register Definitions
b. Refer to the Messaging Unit Chapter for MSI-X Register Definitions.



3.16.3 ATU Vendor ID Register - ATUVID

ATU Vendor ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 136. ATU Vendor ID Register - ATUVID

Internal Bus Address Offset +000H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:00	8086H	ATU Vendor ID - This is a 16-bit value assigned to Intel. This register, combined with the DID, uniquely identify the PCI device. Access type is Read/Write to allow the 81341 and 81342 to configure the register as a different vendor ID to simulate the interface of a standard mechanism currently used by existing application software.

3.16.4 ATU Device ID Register - ATUDID

ATU Device ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 137. ATU Device ID Register - ATUDID

Internal Bus Address Offset +002H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:00	Product Type Dependent ^a	ATU Device ID - This is a 16-bit value assigned to the ATU. This ID, combined with the VID, uniquely identify any PCI device.

^a. See Intel® 81341 and 81342 I/O Processors *Specification Update*.



3.16.5 ATU Command Register - ATUCMD

ATU Command Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3 and in most cases, affect the behavior of the PCI ATU and devices on the PCI Express Link.

Table 138. ATU Command Register - ATUCMD

Internal Bus Address Offset +004H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
15:11	00000 ₂	Reserved
10	0 ₂	Interrupt Disable - Controls the ability of the ATU to generate INTx interrupt messages. When set, the ATU is prevented from generating INTx interrupt messages and generates a Deassert_INTx message for any emulation interrupts already asserted.
09	0 ₂	Fast Back to Back Enable - Does not apply to PCI Express. Hard-wired to 0.
08	0 ₂	SERR# Enable - When set, the ATU is allowed to report non-fatal and fatal error detected by 81341 and 81342 to the Root Complex. Note: Errors are reported either through this bit or through the PCI-Express specific bits in the PCI Express Device Control Register - PE_DCTL .
07	0 ₂	Address/Data Stepping Control - Does not apply to PCI Express. Hard-wired to 0.
06	0 ₂	Parity Error Response - When set, the ATU takes normal action in response to a poisoned TLP received from PCI Express. When cleared, parity checking is disabled. Note: When the bit is cleared but the Poisoned TLP Mask is cleared in the PCI Express Uncorrectable Error Mask - ERRUNC_MSK register, the ATU still logs the error in the Advanced Error Reporting registers and generate an Uncorrectable Error message.
05	0 ₂	VGA Palette Snoop Enable - Does not apply to PCI Express. Hard-wired to 0.
04	0 ₂	Memory Write and Invalidate Enable - Does not apply to PCI Express. Hard-wired to 0.
03	0 ₂	Special Cycle Enable - Does not apply to PCI Express. Hard-wired to 0.
02	0 ₂	Bus Master Enable - When cleared, the ATU is prevented from issuing any memory or I/O read/write requests. Requests other than memory or I/O requests are not controlled by this bit. The ATU initiates a completion transaction regardless of the setting.
01	0 ₂	Memory Enable - Controls the ATU interface's response to memory transactions. When cleared, the ATU drops the transaction and returns an unsupported request
00	0 ₂	I/O Space Enable - Controls the ATU interface response to I/O transactions. When cleared, the ATU drops the transaction and returns an unsupported request.



3.16.6 ATU Status Register - ATUSR

The ATU Status Register bits adhere to the *PCI Local Bus Specification*, Revision 2.3 definitions. The *read/clear* bits can only be set by internal hardware and cleared by either a reset condition or by writing a 1₂ to the register.

Table 139. ATU Status Register - ATUSR

Internal Bus Address Offset +006H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear RO = Read Only PR = Preserved RS = Read/Set NA = Not Accessible		
Bit	Default	Description
15	0 ₂	Detected Parity Error - set when the ATU receives a poisoned TLP regardless of the state of the Parity Error Response in the ATUCMD register.
14	0 ₂	SERR# Asserted - set when the ATU sends an ERR_FATAL or ERR_NONFATAL message, and the SERR Enable bit in the ATUCMD register is '1'.
13	0 ₂	Received Master Abort - set when the ATU receives a completion with Unsupported Request Completion Status.
12	0 ₂	Received Target Abort - set when the ATU receives a completion with Completer Abort Completion Status.
11	0 ₂	Signaled Target Abort - set when the ATU completes a Request using Completer Abort Completion Status.
10:09	00 ₂	DEVSEL# Timing - Does not apply to PCI Express. Hard-wired to 0.
08	0 ₂	Master Data Parity Error - This bit is set by the ATU when its Parity Error Enable bit is set and either of the following two conditions occurs: This bit is set under the following conditions. • ATU receives a Poisoned Completion for an Outbound Read Request • ATU transmits a Poisoned TLP for an Outbound Write Request. When the Parity Error Response bit is cleared in the "ATU Command Register - ATUCMD", this bit is never set.
07	0 ₂	Fast Back-to-Back - Does not apply to PCI Express. Hard-wired to 0.
06	0 ₂	Reserved
05	0 ₂	66 MHz. Capable - Does not apply to PCI Express. Hard-wired to 0.
04	1 ₂	Capabilities List - All PCI Express devices are required to implement the PCI Express capability structure. Hard-wired to 1.
03	0 ₂	Interrupt Status - Indicates that an INTx interrupt message is pending internally to the device. Note: Setting the Interrupt Disable bit to a 1 in (bit 10 of ATUCMD) has no effect on the state of this bit.
02:00	000 ₂	Reserved



3.16.7 ATU Revision ID Register - ATURID

Revision ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 140. ATU Revision ID Register - ATURID

Internal Bus Address Offset +008H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	xxH ^a	ATU Revision - identifies the 81341 and 81342 revision number.

a. See Intel® 81341 and 81342 I/O Processors Specification Update.

3.16.8 ATU Class Code Register - ATUCCR

Class Code Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. Auto configuration software reads this register to determine the PCI device function.

Table 141. ATU Class Code Register - ATUCCR

Internal Bus Address Offset +009H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
23:16	05H	Base Class - Memory Controller
15:08	80H	Sub Class - Other Memory Controller
07:00	00H	Programming Interface - None defined



3.16.9 ATU Cacheline Size Register - ATUCLSR

Cacheline Size Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register is programmed with the system cacheline size in DWORDs (32-bit words). Cacheline Size is restricted to either 0, 8 or 16 DWORDs; the ATU interprets any other value as "0".

Table 142. ATU Cacheline Size Register - ATUCLSR

Internal Bus Address Offset +00CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	ATU Cacheline Size - specifies the system cacheline size in DWORDs. Note: This field is read-write for legacy compatibility purposes but has no impact on any PCI Express device functionality.

3.16.10 ATU Latency Timer Register - ATULT

ATU Latency Timer Register does not apply to PCI Express.

Table 143. ATU Latency Timer Register - ATULT

Internal Bus Address Offset +00DH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	Programmable Latency Timer - The latency timer does not apply to PCI Express. Hard-wired 0.



3.16.11 ATU Header Type Register - ATUHTR

Header Type Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register indicates the layout of ATU configuration space bytes 10H to 3FH. The MSB indicates whether or not the device is multi-function.

Table 144. ATU Header Type Register - ATUHTR

7 4 0 IOP Attributes ro rw rw rw rw rw rw rw PCI Attributes ro ro ro ro ro ro ro ro Internal Bus Address Offset +00EH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07	End Point: DF_SEL[2:0] = "000" Root Complex: 0	Single Function/Multi-Function Device - Identifies the 81341 and 81342 as a single-function or multi-function PCI device depending on the setting of the DF_SEL[2:0] strap during P_RST# assertion. Note: The 81341 and 81342 can be configured as a single-function device (ATU only) or a multi-function device (ATU and storage controller) for split driver support. As a Root Complex this bit is always '0'.
06:00	000000 ₂	PCI Header Type - This bit field indicates the type of PCI header implemented. The ATU interface header conforms to <i>PCI Local Bus Specification</i> , Revision 2.3.



3.16.12 ATU BIST Register - ATUBISTR

The ATU BIST Register controls the functions the Intel XScale® processor performs when BIST is initiated. This register is the interface between the host processor requesting BIST functions and the 81341 and 81342 replying with the results from the software implementation of the BIST functionality.

Table 145. ATU BIST Register - ATUBISTR

Internal Bus Address Offset +00FH Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
07	0 ₂	BIST Capable - This bit value is always equal to the ATUCR ATU BIST Interrupt Enable bit. See Section 3.16.40, "ATU Configuration Register - ATUCR" on page 330 .
06	0 ₂	Start BIST - When the ATUCR BIST Interrupt Enable bit is set: Setting this bit generates an interrupt to the Intel XScale® processor to perform a software BIST function. The Intel XScale® processor clears this bit when the BIST software has completed with the BIST results found in ATUBISTR register bits [3:0]. When the ATUCR BIST Interrupt Enable bit is clear: Setting this bit does not generate an interrupt to the Intel XScale® processor and no BIST functions is performed. The Intel XScale® processor does not clear this bit.
05:04	00 ₂	Reserved
03:00	0000 ₂	BIST Completion Code - when the ATUCR BIST Interrupt Enable bit is set and the ATUBISTR Start BIST bit is set (bit 6): The Intel XScale® processor places the results of the software BIST in these bits. A nonzero value indicates a device-specific error.



The Inbound ATU Base Address Register 0 (IABAR0) together with the Inbound ATU Upper Base Address Register 0 (IAUBAR0) defines the block of memory addresses where the inbound translation window 0 begins. The inbound ATU decodes and forwards the bus request to the 81341 and 81342 internal bus with a translated address to map into 81341 and 81342 local memory. The IABAR0 and IAUBAR0 define the base address and describes the required memory block size; see [Section 3.16.15, “Determining Block Sizes for Base Address Registers” on page 310](#). Bits 31 through 12 of the IABAR0 is either read/write bits or read only with a value of 0 depending on the value located within the IALR0. This configuration allows the IABAR0 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

By default the first 8 Kbytes of memory defined by the IABAR0, IAUBAR0 and the IALR0 is reserved for the Messaging Unit.

Warning: When IALRO is cleared prior to host configuration, the user should also clear the Prefetchable Indicator and the Type Indicator. Assuming IALRO is not cleared:

- Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is cleared prior to host configuration, the user should also set the Type Indicator for 32 bit addressability.
- For compliance to the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability. This is the default for IABAR0.

Table 146. Inbound ATU Base Address Register 0 - IABAR0

Intel® 81341 and 81342 I/O Processors
Developer's Manual
308



3.16.15 Determining Block Sizes for Base Address Registers

The required address size and type can be determined by writing ones to a base address register and reading from the registers. By scanning the returned value from the least-significant bit of the base address registers upwards, the programmer can determine the required address space size. The binary-weighted value of the first non-zero bit found indicates the required amount of space. [Table 148](#) describes the relationship between the values read back and the byte sizes the base address register requires.

Table 148. Memory Block Size Read Response

Response After Writing all 1s to the Base Address Register	Size (in Bytes)	Response After Writing all 1s to the Base Address Register	Size (in Bytes)
FFFFFFF0H	16	FFF00000H	1 M
FFFFFFE0H	32	FFE00000H	2 M
FFFFFFC0H	64	FFC00000H	4 M
FFFFFF80H	128	FF800000H	8 M
FFFFFF00H	256	FF000000H	16 M
FFFFFE00H	512	FE000000H	32 M
FFFFFC00H	1K	FC000000H	64 M
FFFFF800H	2K	F8000000H	128 M
FFFFF000H	4K	F0000000H	256 M
FFFFE000H	8K	E0000000H	512 M
FFFFC000H	16K	C0000000H	1 G
FFFF8000H	32K	80000000H	2 G
FFFF0000H	64K	00000000H	Register not implemented, no address space required.
FFFE0000H	128K		
FFFC0000H	256K		
FFF80000H	512K		

As an example, assume that FFFF.FFFFH is written to the [Inbound ATU Base Address Register 0 - IABAR0](#) and the value read back is FFF0.0008H. Bit zero is a zero, so the device requires memory address space. Bit three is one, so the memory does supports prefetching. Scanning upwards starting at bit four, bit twenty is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicated that the device requires 1 Mbyte of memory space.

The ATU Base Address Registers and the Expansion ROM Base Address Register use their associated limit registers to enable which bits within the base address register are read/write and which bits are read only (0). This allows the programming of these registers in a manner similar to other PCI devices even though the limit is variable.

**Table 149. ATU Base Registers and Associated Limit Registers**

Base Address Register	Limit Register	Description
Inbound ATU Base Address Register 0	Inbound ATU Limit Register 0	Defines the inbound translation window 0 from the PCI Express Link.
Inbound ATU Upper Base Address Register 0	N/A	Together with ATU Base Address Register 0 defines the inbound translation window 0 from the PCI Express Link
Inbound ATU Base Address Register 1	Inbound ATU Limit Register 1	Defines the inbound translation window 1 from the PCI Express Link.
Inbound ATU Upper Base Address Register 1	N/A	Together with ATU Base Address Register 1 defines the inbound translation window 1 from the PCI Express Link
Inbound ATU Base Address Register 2	Inbound ATU Limit Register 2	Defines the inbound translation window 2 from the PCI Express Link.
Inbound ATU Upper Base Address Register 2	N/A	Together with ATU Base Address Register 2 defines the inbound translation window 2 from the PCI Express Link
Expansion ROM Base Address Register	Expansion ROM Limit Register	Defines the window of addresses used by a bus master for reading from an Expansion ROM.



The Inbound ATU Base Address Register 1 (IABAR1) together with the Inbound ATU Upper Base Address Register 1 (IAUBAR1) defines the block of memory addresses where the inbound translation window 1 begins. The inbound ATU decodes and forwards the bus request to the 81341 and 81342 internal bus with a translated address to map into 81341 and 81342 local memory. The IABAR1 and IAUBAR1 define the base address and describes the required memory block size; see [Section 3.16.15, “Determining Block Sizes for Base Address Registers” on page 310](#). Bits 31 through 12 of the IABAR1 is either read/write bits or read only with a value of 0 depending on the value located within the IALR1. This configuration allows the IABAR1 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

- a. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is cleared prior to host configuration, the user should also set the Type Indicator for 32 bit addressability.
- b. For compliance to the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability. This is the default for IABAR1.

Internal Bus Address Offset +018H

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



3.16.17 Inbound ATU Upper Base Address Register 1 - IAUBAR1

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI Express Link for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the type indicator of IABAR1 is set to indicate 32 bit addressability, the IAUBAR1 register attributes are read-only. By default the IAUBAR1 register has read-only attributes. Prior to changing the Type Indicator in the IABAR1 to support 32-bit addressability, the IAUBAR1 must be written with zero unless it already contains zero. Zero is the default value for IAUBAR1.

Table 151. Inbound ATU Upper Base Address Register 1 - IAUBAR1

Bit	Default	Description
31:0	00000H	Translation Upper Base Address 1 - Together with the Translation Base Address 1 these bits define the actual location for this memory window on the PCI Express Link for addresses > 4GBytes.



3.16.19 Inbound ATU Upper Base Address Register 2 - IAUBAR2

This register contains the upper base address when decoding PCI addresses for memory space (Memory Space Indicator in IABAR2 is clear) beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI Express Link for addresses > 4 GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR2 is set to indicate 32 bit addressability or the Memory Space indicator of IABAR2 is set indicating I/O space, the IAUBAR2 register attributes are read-only. By default the IAUBAR2 register has read-only attributes. Prior to changing the Type/Memory Space Indicator in the IABAR2 to support 32-bit addressability, the IAUBAR2 must be written with zero unless it already contains zero. Zero is the default value for IAUBAR2.

Table 153. Inbound ATU Upper Base Address Register 2 - IAUBAR2

		IOP Attributes PCI Attributes Internal Bus Address Offset +024H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																															
31:0	00000H	Translation Upper Base Address 2 - Together with the Translation Base Address 2 these bits define the actual location the translation function is to respond to when addressed from the PCI Express Link for addresses > 4GBytes.																															



3.16.20 ATU Subsystem Vendor ID Register - ASVIR

ATU Subsystem Vendor ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 154. ATU Subsystem Vendor ID Register - ASVIR

1512840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw 1512840 PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +02CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:0	0000H	Subsystem Vendor ID - This register uniquely identifies the add-in board or subsystem vendor.

3.16.21 ATU Subsystem ID Register - ASIR

ATU Subsystem ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 155. ATU Subsystem ID Register - ASIR

1512840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw 1512840 PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +02EH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:0	0000H	Subsystem ID - uniquely identifies the add-in board or subsystem.



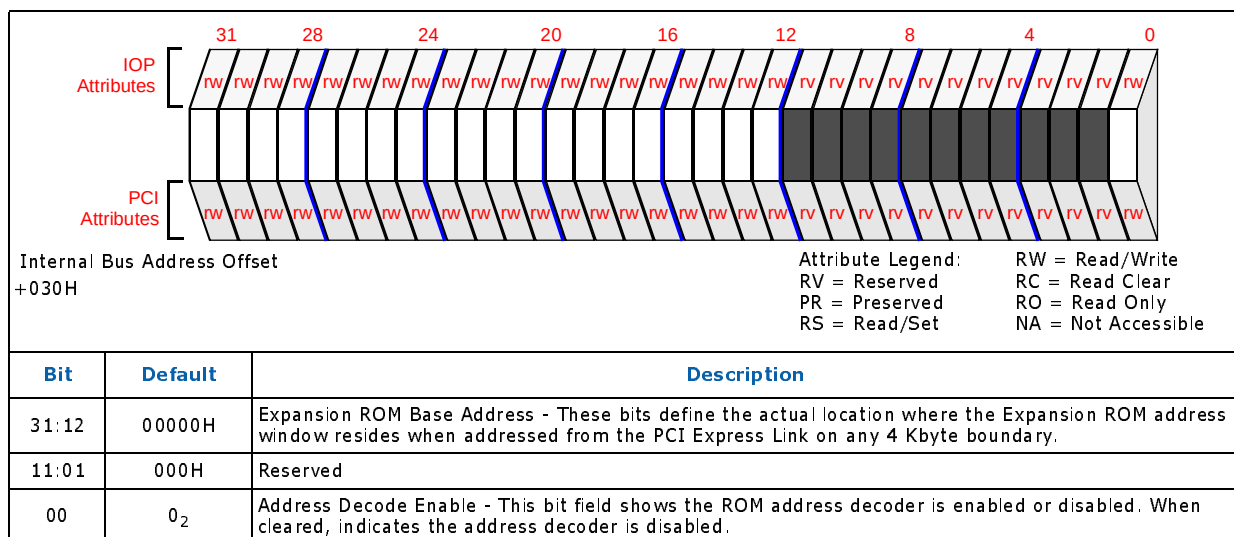
3.16.22 Expansion ROM Base Address Register - ERBAR

The Expansion ROM Base Address Register defines the block of memory addresses used for containing the Expansion ROM. It permits the inclusion of multiple code images, allowing the device to be initialized. The code image supplied consists of either executable code or an interpreted code. Each code image must start on a 512 byte boundary and each must contain the PCI Expansion ROM header. Image placement in ROM space depends on the length of code images which precede it within ROM. ERBAR defines the base address and describes the required memory block size; see [Section 3.16.15](#). Expansion ROM address space (limit size) can be a maximum of 16 MBytes. Bits 31 through 12 of the ERBAR is either read/write bits or read only with a value of 0 depending on the value located within the ERLR. This configuration allows the ERBAR to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The Expansion ROM Base Address Register's programmed value must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming Expansion ROM base address registers.

Note: A write targeting the Expansion ROM window terminates as an Unsupported Request

Table 156. Expansion ROM Base Address Register -ERBAR





3.16.23 ATU Capabilities Pointer Register - ATU_Cap_Ptr

The Capabilities Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register provides an offset in this function's PCI Configuration Space for the location of the first item in the first Capability list. In the case of the 81341 and 81342, this is the PCI Express Link Power Management extended capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 157. ATU Capabilities Pointer Register - ATU_Cap_Ptr

Internal Bus Address Offset +034H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
07:00	98H	Capability List Pointer - This provides an offset in this function's configuration space that points to the 81341 and 81342's PCI Bus Power Management extended capability.



3.16.24 ATU Interrupt Line Register - ATUILR

ATU Interrupt Line Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the system interrupt controller's interrupt request lines which connect to the device's PCI interrupt request lines (as specified in the interrupt pin register).

In a PC environment, for example, the register values and corresponding connections are:

- 0 (00H) through 15 (0FH) correspond to IRQ0 through IRQ15
- 16 (10H) through 254 (FEH) are reserved
- 255 (FFH) indicates "unknown" or "no connection"

Operating system or device driver can examine each device interrupt pin and interrupt line register to determine which system interrupt request line the device uses to issue requests for service.

Table 158. ATU Interrupt Line Register - ATUILR

Internal Bus Address Offset +03CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	FFH	Interrupt Assigned - system-assigned value identifies which system interrupt controller interrupt request line connects to the device's PCI interrupt request lines (as specified in the interrupt pin register). A value of FFH signifies "no connection" or "unknown".



3.16.25 ATU Interrupt Pin Register - ATUIPR

ATU Interrupt Pin Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the interrupt pin the ATU and Messaging Unit interface uses.

Table 159. ATU Interrupt Pin Register - ATUIPR

IOP Attributes PCI Attributes Internal Bus Address Offset +03DH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	01H	Interrupt Used - A value of 01H signifies that the ATU interface unit uses the INTA legacy interrupt message.

3.16.26 ATU Minimum Grant Register - ATUMGNT

This register does not apply to PCI Express.

Table 160. ATU Minimum Grant Register - ATUMGNT

IOP Attributes PCI Attributes Internal Bus Address Offset +03EH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	This register does not apply to PCI Express. Hard-wired to 0



3.16.27 ATU Maximum Latency Register - ATUMLAT

This register does not apply to PCI Express.

Table 161. ATU Maximum Latency Register - ATUMLAT

IOP Attributes PCI Attributes 7 4 0 ro ro ro ro ro ro ro ro Internal Bus Address Offset +03FH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	This register does not apply to PCI Express. Hard-wired to 0



Inbound address translation for memory window 0 occurs for requests originating in the PCI Express domain and targeting the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: Bit 0 can be used to disable claiming of Memory Cycles that hit Inbound Memory Window 0 even though the host processor has allocated memory of the size requested by IABAR0/IALR0[31:12].

Internal Bus Address Offset +040H

Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible

Bit	Default	Description
31:12	FF000H	Inbound Translation Limit 0 - This readback value determines the memory block size required for inbound memory window 0 of the address translation unit. This defaults to an inbound window of 16MB
11:01	000H	Reserved
00	0 ₂	Memory Window 0 Claim Disable -- When clear, Inbound Memory Window 0 claims PCI Memory Cycles Normally. When set, memory transactions targeting Inbound Memory Window 0 is terminated as an Unsupported Request (UR).

3.16.29 Inbound ATU Translate Value Register 0 - IATVR0

The Inbound ATU Translate Value Register 0 (IATVR0) in conjunction with the “[Inbound ATU Upper Translate Value Register 0 - IAUTVR0](#)” on page 323 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 163. Inbound ATU Translate Value Register 0 - IATVR0

Internal Bus Address Offset
+044H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible

3.16.30 Inbound ATU Upper Translate Value Register 0 - IAUTVR0

The Inbound ATU Upper Translate Value Register 0 (IAUTVR0) in conjunction with the "Inbound ATU Translate Value Register 0 - IATVR0" on page 323 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 164. Inbound ATU Upper Translate Value Register 0 - IAUTVR0

Bit	Default	Description
31:04	000 0000H	Reserved
3:0	0H	Inbound Upper ATU Translation Value 0 - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses. The default address allows the ATU to access the internal 81341 and 81342 memory-mapped registers.



Inbound address translation for memory window 1 occurs for transactions originated in the PCI Express domain and targeting the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR1 have a direct effect on the IABAR1 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR1 makes the corresponding bit within the IABAR1 a read only bit which always returns 0. A value of 1 in a bit within the IALR1 makes the corresponding bit within the IABAR1 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR1, all writes to the IABAR1 has no effect since a value of all zeros within the IALR1 makes the IABAR1 a read only register.

Note: Bit 0 can be used to disable claiming of Memory Cycles that hit Inbound Memory Window 1 even though the host processor has allocated memory of the size requested by IABAR1/IALR1[31:12].

Table 165. Inbound ATU Limit Register 1 - IALR1

Internal Bus Address Offset +04CH Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:12	00000H	Inbound Translation Limit 1 - This readback value determines the memory block size required for the ATUs memory window 1.
11:01	000H	Reserved
00	0 ₂	Memory Window 1 Claim Disable -- When clear, Inbound Memory Window 1 claims PCI Memory Cycles Normally. When set, memory transactions targeting Inbound Memory Window 0 is terminated as an Unsupported Request (UR).



3.16.32 Inbound ATU Translate Value Register 1 - IATVR1

The Inbound ATU Translate Value Register 1 (IATVR1) in conjunction with the “Inbound ATU Upper Translate Value Register 1 - IAUTVR1” on page 325 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the Inbound ATU address translation.

Table 166. Inbound ATU Translate Value Register 1 - IATVR1

		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
Bit	Default	Description																											
31:12	00000H	Inbound ATU Translation Value 1 - Bits 31 to 12 of the internal bus address used to convert PCI address to internal bus addresses. This value must be naturally aligned with the IABAR1 register's programmed value (see Section 3.16.15, “Determining Block Sizes for Base Address Registers” on page 310).																											
11:01	000H	Reserved																											
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read/write transactions through BAR1. When clear, no swap is performed. Refer to Section 3.4, “Big Endian Byte Swapping” on page 259 for more details.																											

3.16.33 Inbound ATU Upper Translate Value Register 1 - IAUTVR1

The Inbound ATU Upper Translate Value Register 1 (IAUTVR1) in conjunction with the “Inbound ATU Translate Value Register 1 - IATVR1” on page 325 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 167. Inbound ATU Upper Translate Value Register 1 - IAUTVR1

		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
Bit	Default	Description																											
31:04	000 0000H	Reserved																											
3:0	0H	Inbound Upper ATU Translation Value 1 - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.																											



Inbound address translation for inbound window 2 occurs for transactions originated in the PCI Express domain and targeting the 81341 and 81342 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The 81341 and 81342 value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 81341 and 81342 value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 8 within the IALR2 have a direct effect on the IABAR2 register, bits 31 to 8, with a one to one correspondence. A value of 0 in a bit within the IALR2 makes the corresponding bit within the IABAR2 a read only bit which always returns 0. A value of 1 in a bit within the IALR2 makes the corresponding bit within the IABAR2 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR2, all writes to the IABAR2 has no effect since a value of all zeros within the IALR2 makes the IABAR2 a read only register.

Note: Bit 0 can be used to disable claiming of PCI Cycles that hit Inbound Window 1 even though the host processor has allocated memory of the size requested by IABAR2/IALR2[31:8].

Internal Bus Address Offset +058H Attribute Legend: RV = Reserved RC = Read Clear RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:8	00000H	Inbound Translation Limit 2 - This readback value determines the memory block size required for the ATUs memory window 2.
07:01	000H	Reserved
00	0 ₂	Window 2 Claim Disable -- When clear, Inbound Memory Window 2 claims PCI Memory or I/O Cycles Normally. When set, transactions targeting Inbound Memory Window 2 is terminated as an Unsupported Request (UR).



3.16.35 Inbound ATU Translate Value Register 2 - IATVR2

The Inbound ATU Translate Value Register 2 (IATVR2) in conjunction with the [“Inbound ATU Upper Translate Value Register 2 - IAUTVR2” on page 328](#) contain bits 35 to 8 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the Inbound ATU address translation.

Warning: When the IABAR2 register's Memory Space Indicator is set, inbound window 2 will be in I/O space. In this case, the *PCI Local Bus Specification*, Revision 2.3 requires that the IABAR2 request no more than 256 bytes of I/O space. Thus, IALR2 must be set to FFFF FF00H when the Memory Space Indicator in IABAR2 is set.

Warning: Although IATVR2[09:08] are programmable bits, hardware does not translate these bits for PCI windows that are defined to be less than 1-Kbyte. Hardware uses the PCI Address bits[09:08] as captured on the PCI address bus to drive the internal bus address[09:08].

Table 169. Inbound ATU Translate Value Register 2 - IATVR2

Internal Bus Address Offset +05CH

Attribute Legend:

- RW = Read/Write
- RV = Reserved
- PR = Preserved
- RS = Read/Set
- RC = Read Clear
- RO = Read Only
- NA = Not Accessible



The Inbound ATU Upper Translate Value Register 2 (IAUTVR2) in conjunction with the “Inbound ATU Translate Value Register 2 - IATVR2” on page 327 contain bits 35 to 8 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Internal Bus Address Offset +060H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:04	000 0000H	Reserved
3:0	0H	Inbound Upper ATU Translation Value 2 - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.

The Expansion ROM Limit Register (ERLR) defines the block size of addresses the ATU defines as Expansion ROM address space. Block size is programmed by writing a value into the ERLR.

Bits 31 to 12 within the ERLR have a direct effect on the ERBAR register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the ERLR makes the corresponding bit within the ERBAR a read only bit which always returns 0. A value of 1 in a bit within the ERLR makes the corresponding bit within the ERBAR read/write from PCI.

Internal Bus Address Offset +064H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 RC = Read Clear
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



3.16.38 Expansion ROM Translate Value Register - ERTVR

The Expansion ROM Translate Value Register 0 (ERTVR) in conjunction with the “Expansion ROM Upper Translate Value Register - ERUTVR” on page 329 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the Expansion ROM address translation.

Table 172. Expansion ROM Translate Value Register - ERTVR

Bit	Default	Description																													
31:12	00000H	Expansion ROM Translation Value - This value represents bits 31 to 12 of the internal bus address used to convert the PCI address to internal bus addresses. This value must be naturally aligned with the ERBAR register's programmed value (see Section 3.16.15, “Determining Block Sizes for Base Address Registers” on page 310).																													
11:01	000H	Reserved																													
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read transactions through ERBAR. When clear, no swap is performed. Refer to Section 3.4, “Big Endian Byte Swapping” on page 259 for more details.																													

3.16.39 Expansion ROM Upper Translate Value Register - ERUTVR

The Expansion ROM Upper Translate Value Register (ERUTVR) in conjunction with the “Expansion ROM Translate Value Register - ERTVR” on page 329 contain bits 35 to 12 of the internal bus address used to convert PCI Express Link addresses. The converted address is driven on the internal bus as a result of the Expansion ROM address translation.

Table 173. Expansion ROM Upper Translate Value Register - ERUTVR

Bit	Default	Description																													
31:04	000 0000H	Reserved																													
3:0	0H	Expansion ROM Upper Translation Value - This value represents bits 35 to 32 of the internal bus address used to convert the PCI address to internal bus addresses.																													



3.16.40 ATU Configuration Register - ATUCR

The ATU Configuration Register contains some additional parameters in the ATU.

Table 174. ATU Configuration Register - ATUCR

Internal Bus Address Offset +070H		
Attribute Legend: RW = Read/Write, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31	0	Reserved
30	0	Completion Timeout Disable. - When set, this bit disables the completion timeout mechanism for outbound non-posted requests. Read/Write.
29:09	0	Reserved
08	0	Halt on Error. - When enabled, an Address or Data Parity an Outbound write request results in the following actions: - Stop accepting outbound requests by clearing the outbound enable. - Flush any pending outbound write requests from the transaction queue that are behind the one that had the error. - Assert the Halt On Error Interrupt to the Intel XScale® Processor. - Assert the appropriate error interrupt to the Intel XScale® Processor. - Read requests is allowed to proceed normally though no new ones is accepted.
07	0	Reserved
06	1	Drop subsequent inbound vendor defined messages (IVM). 0 = When cleared, subsequent IVMs remains in the inbound posted queue and blocks all other transactions. 1 = When set and the inbound vendor defined message registers are full with a pending IVM, subsequent IVMs is discarded. This is necessary to prevent a deadlock condition when the Intel XScale® processor needs to receive a read completion before it can handle the IVM interrupt.
05	0	Outbound Completion Size - This bit controls how completions are returned on the PCI Express interface. 0 = Max Payload Size setting is used to format completions. Once enough data has accumulated to reach a Max Payload address boundary, the completion TLP is formed and pushed into the completion queue. Completion TLPs has a payload size of 128, 256, or 512 bytes depending on the setting of the Max_Payload_Size field in the "PCI Express Device Control Register - PE_DCTL" on page 348. 1 = 128B Payload Size. All completions uses a maximum size of 128 bytes regardless of the setting of the Max_Payload_Size field.
04	0	Inbound Minimum Completion Size - This bit controls how completion data is returned to the internal bus. 0 = Return completion data as it is received from the PCI Express interface. This size may be as small as 64Bytes. 1 = Wait for the entire request to be satisfied before returning completion data.
03	0	ATU BIST Interrupt Enable - When set, enables an interrupt to the Intel XScale® processor when the start BIST bit is set in the ATUBISTR register. This bit is also reflected as the BIST Capable bit 7 in the ATUBISTR register.
02	0 ₂	Reserved
01	0 ₂	Outbound ATU Enable - When set, enables the outbound address translation unit. When cleared, disables the outbound ATU.
00	0 ₂	Reserved



3.16.41 PCI Configuration and Status Register - PCSR

The PCI Configuration and Status Register has additional bits for controlling and monitoring various features of the PCI Express interface.

Warning: The PCI Express Bus Number and Device Number are used to form the Requestor/Completer ID and should only be changed when operating as a Root Complex. These fields are updated whenever a type 0 configuration write targets the IOP. System instability may result when the Bus/Device numbers are modified while operating as an endpoint.

Table 175. PCI Configuration and Status Register - PCSR (Sheet 1 of 2)

IOP Attributes PCI Attributes																																	
		Internal Bus Address Offset +074H																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only CO = Clear Only															
Bit	Default	Description																															
31:24	00H	PCI Express Bus Number																															
23:19	0_0000 ₂	PCI Express Device Number																															
18:16	000 ₂	PCI Express Function Number																															
15	0 ₂	Outbound Transaction Queue Busy: 0 = Outbound Transaction Queue Empty 1 = Outbound Transaction Queue Busy Note: This tracks outbound transactions and includes the Outbound Non-Posted, Outbound Posted, and Inbound Completion queues.																															
14	0 ₂	Inbound Transaction Queue Busy: 0 = Inbound Transaction Queue Empty 1 = Inbound Transaction Queue Busy Note: This tracks inbound transactions and includes the Inbound Non-Posted, Inbound Posted, and Outbound Completions queues.																															
13	Varies with external state of the PCIE_RC# strap.	PCI Express Root Complex mode 0 = PCI Express Root Complex enabled 1 = PCI Express Root Complex disabled (end point mode).																															
12	0	Link Layer Retry Buffer (LLRB) Busy. 0 = LLRB Empty 1 = LLRB Busy																															
11:10	00 ₂	Reserved																															



Table 175. PCI Configuration and Status Register - PCSR (Sheet 2 of 2)

		Internal Bus Address Offset +074H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set CO = Clear Only																															
Bit	Default	Description																															
9:8	00	Core reset. These bits are used to place the Intel XScale® microarchitecture processors into reset. When a '1' is written to these bits the corresponding Core Processor Reset bit (Bits1:0) is set. These bits are self clearing and always reads '0'.																															
07	Varies with external state of the FW_TIMER_OFF# strap	Firmware Timer Disable 0 = Firmware Timeout is disabled. 1 = Firmware Timeout is enabled When enabled, a 400mS timer is started at the trailing edge of reset. The Configuration Request Retry (bit 2 of this register) is cleared when it is not cleared before the expiration of the timer. For example, when firmware does not clear the Configuration Request Retry bit before the timer expires. The firmware timer bit is also automatically cleared by the firmware timer when the Configuration Request Retry bit is still set when the timer expires. After the host is allowed access, the firmware timer bit can be used to indicate how the Configuration Request Retry bit was cleared: 0 = The firmware timer expired and cleared both the Configuration Request Retry and the firmware timer bits. 1 = Firmware cleared the Configuration Request Retry bit before the timer expired. Note: When the firmware timer is disabled, firmware is responsible to clear the Configuration Request Retry bit. Otherwise, the ATU indefinitely retries all host configuration cycles.																															
05:03	00 ₂	Reserved																															
02	Varies with external state of RETRY pin at PCI Express Link reset	Configuration Request Retry - When this bit is set, the PCI Express interface of the 81341 and 81342 responds to all configuration cycles with a Completion Retry Status (CRS) condition. When clear, the 81341 and 81342 responds to the appropriate configuration cycles. The default condition for this bit is based on the external state of the RETRY pin at the rising edge of P_RST#. When the external state of the pin is high, the bit is set. When the external state of the pin is low, the bit is cleared. Note: This bit self-clears when the Firmware Timer expires. Bit 7 of this register is used to stop the firmware timer.																															
01:00	Varies with external state of RST_MODE[1:0]# pin at PCI Express Link reset	Core Processor Reset - This bit is set to its default value by the hardware when either P_RST# is asserted or the Reset Internal Bus bit in PCSR is set. When this bit is set, the Intel XScale® processor is being held in reset. Software cannot set this bit. Software is required to clear this bit to deassert Intel XScale® processor reset. The default condition for this bit is based on the external state of the RST_MODE[1:0]# pin at the rising edge of P_RST#. When the external state of the pin is low, the default value of this bit is set. When the external state of the pin is high, the default value of this bit is clear. Note: This bits are "Clear Only". A write of '0' clears the bit, a write of '1' has no affect.																															



3.16.42 ATU Interrupt Status Register - ATUISR

The ATU Interrupt Status Register is used to notify the core processor of the source of an ATU interrupt. In addition, this register is written to clear the source of the interrupt to the interrupt unit of the 81341 and 81342. All bits in this register are Read/Clear.

Bits 4:0 are a direct reflection of bits 15, 13:11, and bit 8 (respectively) of the ATU Status Register (these bits are set at the same time by hardware but need to be cleared independently). Bit 5 is set by an error associated with the internal bus of the 81341 and 81342. Bit 24 is for software BIST. The conditions that result in an ATU interrupt are cleared by writing a 1 to the appropriate bits in this register.

Note: The interrupt status bits are not set when the corresponding mask bit is set.

Table 176. ATU Interrupt Status Register - ATUISR (Sheet 1 of 3)

31 28 24 20 16 12 8 4 0 IOP Attributes PCI Attributes		Internal Bus Address Offset +078H Attribute Legend: RZ = Reserved Zero PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description	
31:29	0	Reserved Zero - Software must write 0 to these bits.	
28	0	Slot Power Message Received - Indicates a Set_Slot_Power_Limit Message was received and logged in the "PCI Express Device Capabilities Register - PCIE_DCAP" on page 347. Generates the ATU Inbound Message Interrupt.	
27	0	PME Interrupt - Indicates a PME message was received and logged in the "PCI Express Root Status Register PE_RSR" on page 358. Note: This read only bit is a copy of the start PME Status bit in the PE_RSR register. To clear this interrupt software must clear the PME Status bit. The "mask" for this interrupt is controlled by the PME interrupt enable in the "PCI Express Root Control Register - PE_RCR" on page 357. Generates the ATU Inbound Message Interrupt	
26	0	Hot-Plug Message Received - This bit set when a Hot-Plug message is received that changes the value of the Attention Indicator Status or Power Indicator Status in the "PCI Express Message Control and Status Register - PEMCSR". Generates the ATU Inbound Message Interrupt	
25	0	Inbound Vendor Message Received (IVM) - Note while this bit is asserted, any additional Inbound Vendor Messages may be blocked which can block all other inbound transactions. This bit must be cleared after the message has been processed. See Section 3.3.1.6, "Inbound Vendor_Defined Message Transactions" on page 248 for more details. Generates the ATU Inbound Message Interrupt	
24	0	ATU BIST Interrupt - When set, the host processor has set the start BIST, ATUBISTR register bit 6, and the ATU BIST interrupt enable (ATUCR register bit 3) is enabled. The Intel XScale® processor can initiate the software BIST and store the result in ATUBISTR register bits 3:0. Note: This read only bit is a copy of the start BIST bit in the ATUBISTR register. To clear this interrupt software must clear the start BIST bit. The "mask" for this interrupt is controlled by the BIST interrupt enable in the ATUCR register. Generates the ATU BIST Interrupt	
23:19	0	Reserved Zero - Software must write 0 to these bits.	
18	0	ATU Configuration Write - This bit is set when a PCI Express configuration write occurs to any enabled function. When set, this bit results in the assertion of the ATU Config Reg Write Interrupt.	



Table 176. ATU Interrupt Status Register - ATUISR (Sheet 2 of 3)

Internal Bus Address Offset +078H		Attribute Legend: RZ = Reserved Zero PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
IOP Attributes		PCI Attributes	
31 28 24 20 16 12 8 4 0		RZ RZ RZ RC RC RC RO RZ RZ RZ RZ RZ RC RC RC RZ RZ RC RC RC RO RO RC RC RC RC RC RC RC RC RC RC	
Bit	Default	Description	
17	0	VPD Address Register Updated - Set when a configuration write occurs to the VPDAR register. When set, this bit results in the assertion of the ATU Config Reg Write Interrupt.	
16	0	Power State Transition - Set when the Power State Field of the ATU Power Management Control/Status Register is written to transition from D0 ->D3, D0 ->D1, D1 -> D3, or D3 ->D0. When set, this bit results in the assertion of the ATU Config Req Write Interrupt.	
15:14	0	Reserved Zero - Software must write 0 to these bits	
13	0	Halt on Error Interrupt - Set when an outbound address or data parity error is detected on a posted write and the Halt on Error control (bit 8) is set in the "ATU Configuration Register - ATUCR"	
12	0	Root System Error Interrupt - This bit set when the System Error enable bits are set in the "PCI Express Root Control Register - PE_RCR" on page 357 and a non-masked ERR_FATAL, ERR_NONFATAL, or ERR_COR message is received. Note: A PCI Express Error message may also cause bit 11 to be set. Generates the ATU Error Interrupt	
11	0	Root Error Message received - Indicates a Root Complex error message was received and logged in the "Root Error Status Register - RERR_SR" Note: This read only bit is set when any of bits 6, 5, or 0 is set in the RERR_SR registers. To clear this interrupt, these bit must be cleared in the RERR_SR register. The mask for this interrupt is controlled by the "Root Error Command Register - RERR_CMD" register. Generates the ATU Error Interrupt	
10	0	PCI Interface Error - Indicates a non-masked error was logged in the "PCI Interface Error Status - PIE_STS" register. Note: This read only bit is an OR of the unmasked PIE_STS bits. Generates the ATU Error Interrupt	
09	0	Correctable Error Message Transmitted - Indicates a ERR_COR message was sent to the Root Complex. A message is transmitted when a correctable error was logged and the corresponding mask is cleared in the "PCI Express Correctable Error Mask - ERRCOR_MSK" register Generates the ATU Error Interrupt	
08	0	Uncorrectable Error Message Transmitted - Indicates an ERR_FATAL or ERR_NONFATAL message was sent to the root complex. A message is transmitted when an uncorrectable error was logged and the corresponding mask is cleared in the "PCI Express Uncorrectable Error Mask - ERRUNC_MSK" register Generates the ATU Error Interrupt	
07	0	Received Configuration Retry Status (CRS) - This bit set whenever an outbound configuration request is completed with a CRS status. Generates the ATU Error Interrupt	
06	0	Link Down Interrupt - This bit is set whenever the PCI Express link goes down.	
05	0	Internal Bus Master Abort - Set when a transaction initiated by the ATU internal bus initiator interface ends in a Master-abort. Note: An internal bus master abort may also result in the setting of the Signaled Target Abort. Generates the ATU Error Interrupt	
04	0	Detected Parity Error Interrupt - Set when a poisoned TLP is received by any function, regardless of the state of the Parity Error Response. Generates the ATU Error Interrupt	



3.16.43 ATU Interrupt Mask Register - ATUIMR

The ATU Interrupt Mask Register contains the control bit to enable and disable interrupts generated by the ATU.

Table 177. ATU Interrupt Mask Register - ATUIMR

Internal Bus Address Offset +07CH																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																	
Bit	Default	Description																															
31:29	0	Reserved																															
28	0	Slot Power Message Received Interrupt Mask - When '1' the interrupt is masked.																															
27	0	Reserved Note: The PME interrupt is masked in the "PCI Express Root Control Register - PE_RCR" on page 357																															
26	0	Hot-Plug Message Received Interrupt Mask - When '1' the interrupt is masked																															
25	0	Inbound Vendor Message Received Interrupt Mask - When '1' the interrupt is masked																															
24	0	Reserved Note: The ATUBIST interrupt is masked by the BIST interrupt enable in the ATUCR register.																															
23:19	0	Reserved																															
18	0	ATU Configuration Write Mask - When '1' the interrupt is masked.																															
17	0	VPD Address Register Updated Mask- When '1' the interrupt is masked																															
16	0	Power State Transition Mask - When '1' the interrupt is masked																															
15:14	0	Reserved																															
13	0	Halt on Error Interrupt Mask - When '1' the interrupt is masked.																															
12	0	Root System Error Interrupt Mask - When '1' the interrupt is masked																															
11	0	Reserved																															
10	0	Reserved Note: ATUISR[10] is controlled by the "PCI Interface Error Mask - PIE_MSK" register.																															
09	0	Correctable Error Logged Interrupt Mask - When '1' the interrupt is masked																															
08	0	Uncorrectable Error Logged Interrupt Mask - When '1' the interrupt is masked																															
07	0	Received Configuration Retry Status (CRS) Mask - When '1' the interrupt is masked																															
06	0	Link Down Interrupt Mask - When '1' the interrupt is masked.																															
05	0	Internal Bus Master Abort Interrupt Mask - When '1' the interrupt is masked																															
04	0	Data Parity Error Interrupt Mask - When '1' the interrupt is masked																															
03	0	Received Master Abort Interrupt Mask - When '1' the interrupt is masked																															
02	0	Signaled Target Abort Interrupt Mask - When '1' the interrupt is masked																															
01	0	Received Target Abort Interrupt Mask - When '1' the interrupt is masked																															
00	0	Master Data Parity Error Interrupt Mask - When '1' the interrupt is masked																															



3.16.44 PCI Express Message Control/Status Register - PEMCSR

The PCI Express Message Control/Status Register controls the generation and logs the receipt of PCI Express Power Management and Hot-Plug messages.

Table 178. PCI Express Message Control and Status Register - PEMCSR

IOP Attributes PCI Attributes 31 28 24 20 16 12 8 4 0 ro ro ro ro rv	



3.16.45 PCI Express Link Control/Status Register - PELCSR

The PCI Express Link Control/Status Register controls various parameters of the Link Layer including Training Sequence.

Note: When operating as an endpoint, these bits operate as status bits and reflect the settings of the most recent TS1/TS2 training sequences. When operating as a Root Complex, these bits are control bits that are used when sending the training sequences. Training sequences can be initiated by setting the Retrain Link bit in the “PCI Express Link Control Register PE_LCTL” on page 352

Table 179. PCI Express Link Control and Status Register - PELCSR

Internal Bus Address Offset +084H																		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible															
Bit	Default	Description																															
31:6	0000H	Reserved																															
5		Scrambling Disabled Status 0 = Scrambling Active 1 = Scrambling Disabled																															
4		Loopback Status 0 = Loopback Disabled 1 = Loopback Enabled																															
3	0	Disable Scrambling - This controls the disable scrambling bit in the TS1 Ordered Set. 0 = Scrambling Active 1 = Scrambling Disabled																															
2	0	Loopback - This controls the loopback bit in the TS1 Ordered Set. 0 = Loopback Disabled 1 = Loopback Enabled																															
1	0	Reserved Note: The Disable Link control is located in the “PCI Express Link Control Register PE_LCTL” on page 352																															
0	0	Hot Reset - When operating as an endpoint, this is a status bit. When operating as a Root Complex, this is a control bit. 0 = Hot Reset Disabled 1 = Hot Reset Enabled																															



3.16.46 VPD Capability Identifier Register - VPD_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the VPD extended capability with an ID of 03H as defined by the *PCI Local Bus Specification*, Revision 2.3.

Table 180. VPD Capability Identifier Register - VPD_Cap_ID

Internal Bus Address Offset +090H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	03H	Cap_Id - This field with its' 03H value identifies this item in the linked list of Extended Capability Headers as being the VPD capability registers.

3.16.47 VPD Next Item Pointer Register - VPD_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 81341 and 81342, this the final capability list, and hence, this register is set to 00H.

Table 181. VPD Next Item Pointer Register - VPD_Next_Item_Ptr

Internal Bus Address Offset +091H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list. Since the VPD capabilities are the last in the linked list of extended capabilities in the 81341 and 81342, the register is set to 00H.



3.16.48 VPD Address Register - VPDAR

The VPD Address register (VPDAR) contains the DWORD-aligned byte address of the VPD to be accessed. The register is read/write and the initial value at power-up is indeterminate.

A PCI Configuration Write to the VPDAR interrupts the Intel XScale® processor. Software can use the Flag setting to determine whether the configuration write was intended to initiate a read or write of the VPD through the VPD Data Register.

Table 182. VPD Address Register - VPDAR

Internal Bus Address Offset +092 Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15	0 ₂	Flag - A flag is used to indicate when a transfer of data between the VPD Data Register and the storage component has completed. Please see Section 3.13, "Vital Product Data" on page 286 for more details on how the 81341 and 81342 handles the data transfer.
14:0	0000H	VPD Address - This register is written to set the DWORD-aligned byte address used to read or write Vital Product Data from the VPD storage component.

3.16.49 VPD Data Register - VPDDR

This register is used to transfer data between the 81341 and 81342 and the VPD storage component.

Table 183. VPD Data Register - VPDDR

Internal Bus Address Offset +094H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:0	0000H	VPD Data - Four bytes are always read or written through this register to/from the VPD storage component.



3.16.50 PM Capability Identifier Register - PM_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the PCI Express Link Power Management extended capability with an ID of 01H as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 184. PM_Capability Identifier Register - PM_Cap_ID

Internal Bus Address Offset +098H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	01H	Cap_Id - This field with its' 01H value identifies this item in the linked list of Extended Capability Headers as being the PCI Power Management Registers.

3.16.51 PM Next Item Pointer Register - PM_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 81341 and 81342, the next capability (MSI-X capability list) is located at off-set B0H. Note that the PM_Next_Item_Ptr can be written by the processor.

Table 185. PM Next Item Pointer Register - PM_Next_Item_Ptr

Internal Bus Address Offset +099H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	B0H	Next_Item_Pointer - Provides an offset into the function's configuration space pointing to the next item in the function's capability list which in the 81341 and 81342 is the MSI-X extended capabilities header.



3.16.52 ATU Power Management Capabilities Register - APMCR

Power Management Capabilities bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This register is a 16-bit read-only register which provides information on the capabilities of the ATU function related to power management.

Table 186. ATU Power Management Capabilities Register - APMCR

15 12 8 4 0 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro pr ro ro ro ro PCI Attributes ro ro ro ro ro ro ro ro ro ro ro pr ro ro ro ro Internal Bus Address Offset +09AH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:11	0000 ₂	PME_Support - Not capable of asserting the PME# signal in any state, since PME# is not supported by the 81341 and 81342.
10	0 ₂	D2_Support - Set to 0 ₂ indicating the 81341 and 81342 does not support the D2 Power Management State
9	1 ₂	D1_Support - Set to 1 ₂ indicating that the 81341 and 81342 supports the D1 Power Management State
8:6	000 ₂	Aux_Current - This field is set to 000 ₂ indicating that the 81341 and 81342 has no current requirements for the 3.3Vaux signal as defined in the <i>PCI Bus Power Management Interface Specification</i> , Revision 1.1
5	0 ₂	DSI - Set to 0 ₂ meaning that this function does not require a device specific initialization sequence following the transition to the D0 uninitialized state.
4	0 ₂	Preserved.
3	0 ₂	PME Clock - Does not apply to PCI Express. Hard-wired 0
2:0	010 ₂	Version - Setting these bits to 010 ₂ means that this function complies with <i>PCI Bus Power Management Interface Specification</i> , Revision 1.1



3.16.53 ATU Power Management Control/Status Register - APMCSR

Power Management Control/Status bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This 16-bit register is the control and status interface for the power management extended capability.

Note: Some bits in this register are sticky through reset.

Table 187. ATU Power Management Control/Status Register - APMCSR

Internal Bus Address Offset +09CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15	0 ₂	PME_Status - This function is not capable of asserting the PME# signal in any state, since PME## is not supported by the 81341 and 81342. Hard-wired 0
14:9	00H	Reserved
8	0 ₂	PME_En - This bit is hard-wired to read-only 0 ₂ since this function does not support PME# generation from any power state.
7:2	000000 ₂	Reserved
1:0	00 ₂	Power State - This 2-bit field is used both to determine the current power state of a function and to set the function into a new power state. The definition of the values is: 00₂ - D0 01₂ - D1 10₂ - D2 (Unsupported) 11₂ - D3_{hot} The 81341 and 81342 supports the D0, D1, and D3_{hot} states. Note: A write of 10 to this field is discarded and does not change to power state. Additionally a change of state from D0->D3, D0->D1, D1->D3, or D3->D0 can result in setting bit 16 of the ATUISR.



3.16.54 ATU Scratch Pad Register - ATUSPR

This register can be used for application specific purposes and has no direct impact on the hardware.

Table 188. Scratch Pad Register - ATUSPR

		Internal Bus Address Offset +0CCH																																			
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																																			
Bit	Default	Description																																			
31:0	0000H	Scratch Pad Data - Entire register is available for application specific purposes.																																			

3.16.55 PCI Express Capability List Register - PCIE_CAPID

The Capability Identifier Register bits adhere to the definitions in the *PCI Express Base Specification*, Revision 1.0a. This is the PCI Express Capability List with an ID of 10H as defined by the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0.

Table 189. PCI Express Capability Identifier Register - PCIE_CAPID

		Internal Bus Address Offset +0D0H							
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible							
Bit	Default	Description							
07:00	10H	Cap_Id - This field with its' 10H value identifies this item in the linked list of Extended Capability Headers as being the PCI Express Capability List registers.							



3.16.56 PCI Express Next Item Pointer Register - PCIE_NXTP

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list.

By default, the PCI Express capability is the last capabilities list for the 81341 and 81342, thus this register defaults to 00H.

However, this register may be written to 90H prior to host configuration to include the VPD capability located at off-set 90H.

Warning: Writing this register to any value other than 00H (default) or 90H is not supported and may produce unpredictable system behavior.

In order to insure that this register is written prior to host configuration, the 81341 and 81342 must be initialized at **P_RST#** assertion to Retry Type 0 configuration cycles (bit 2 of PCSR). Typically, the Intel XScale® processor would be enabled to boot immediately following **P_RST#** assertion in this case (bit 1 of PCSR), as well. Please see [Section 3.16.41, "PCI Configuration and Status Register - PCSR"](#) on page 331 for more details on the 81341 and 81342's initialization modes.

Table 190. PCI Express Next Item Pointer Register - PCIE_NXTP

Internal Bus Address Offset +0D1H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	00H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list. Since the PCI Express capabilities are the last in the linked list of extended capabilities in the 81341 and 81342, the register is set to 00H. However, this field may be written prior to host configuration with 90H to extend the list to include the VPD extended capabilities header.



3.16.57 PCI Express Capabilities Register - PCIE_CAP

This register controls various modes and features of ATU and Message Unit when operating in the PCI Express mode.

Table 191. PCI Express Capabilities Register PCIE_CAP

Internal Bus Address Offset +0D2H 15 12 8 4 0 IOP Attributes pr pr ro ro ro ro ro ro ro ro ro ro ro ro ro ro PCI Attributes pr pr ro ro ro ro ro ro ro ro ro ro ro ro ro ro Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:14	00 ₂	Preserved
13:9	00000	Interrupt Message Number - 81341 and 81342 uses a single interrupt message. This register is hardcoded to 0.
8	Endpoint: 0 Root Complex: 1	Slot Implemented - Indicates that the PCI Express Link associated with this port is connected to a slot. Only valid for root complex and switch downstream ports. Hard-wired to 0 Set to 1 for root complex this bit is initialized via strapping options.
7:4	0000 or 0100	Device/Port Type - Indicates the type of PCI Express logical device. 0000b - PCI Express Endpoint device 0100b - Root Port of PCI Express Root Complex these bits are initialized via strapping options.
3:0	1H	Capability Version - Indicates PCI-SIG defined PCI Express capability structure version number 81341 and 81342 supports version 1h.



3.16.58 PCI Express Device Capabilities Register - PCIE_DCAP

This register identifies the capabilities and current operating mode of ATU, DMAs and Message Unit when operating in the PCI Express mode.

Table 192. PCI Express Device Capabilities Register - PCIE_DCAP

31 28 24 20 16 12 8 4 0 IOP Attributes pr pr pr pr ro ro ro ro ro ro ro ro ro ro pr pr rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw PCI Attributes pr pr pr pr ro ro ro ro ro ro ro ro ro ro pr pr ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +0D4H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:28	00 ₂	Preserved
27:26	00	Captured Slot Power Limit Scale - Specifies the scale used for the Slot Power Limit Value. This value is set when the Set_Slot_Power_Limit message is received.
25:18	00H	Captured Slot Power Limit Value - In combination with the Slot Power Limit Scale value, specifies the upper limit on power supplied by slot. This value is set when the Set_Slot_Power_Limit message is received.
17:16	00	Preserved
15	1	Role-Based Error Reporting - this bit is set to indicate that this device implements the Role Base Error Reporting defined in <i>PCI Express Base Specification</i> , Revision 1.1.
14	0	Power Indicator Present on Device - When set, indicates that a Power Indicator is implemented on the card or module.
13	0	Attention Indicator Preset on Device - When set, indicates that an Attention Indicator is implemented on the card or module
12	0	Attention Button Present on Device - When set, indicates that an Attention Button is implemented on the card or module
11:9	000	Endpoint L1 Acceptable Latency - 81341 and 81342 does not support L1 active state power management.
8:6	111	Endpoint L0 Acceptable Latency -Total acceptable latency that 81341 and 81342 can withstand due to a transition from L0s to L0 state.
5	0	Extended Tag Field Supported - Indicates the maximum supported size of the Tag field as a Requester. 81341 and 81342 does not generate 8-bit Tags but supports 8-bit Tags as a completer.
4:3	00	Phantom Functions Supported The ATU does not use phantom functions to extend the number of outstanding requests.
2:0	010	Max Payload Size Supported - Indicates that 81341 and 81342 can support a max payload of 512B



3.16.59 PCI Express Device Control Register - PE_DCTL

This register controls various modes and features of ATU and Message Unit when operating in the PCI Express mode.

Table 193. PCI Express Device Control Register - PE_DCTL (Sheet 1 of 2)

Internal Bus Address Offset +0D8H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15	0 ₂	Preserved
14:12	010 ₂	Max_Read_Request_Size – This field sets the maximum Read Request size for the Device as a Requester. The Device must not generate read requests with size exceeding the set value. Defined encodings for this field are: 000b 128B max read request size 001b 256B max read request size 010b 512B max read request size 011b 1024B max read request size 100b 2048B max read request size 101b 4096B max read request size 110b Reserved 111b Reserved Any reserved value is treated as 4096B. Note: In a multifunction configuration, the minimum programmed value from all functions is used when issuing requests.
11	1	Enable No Snoop
10	0	Aux Power PM Enable - The ATU does not utilize Auxiliary power. Hard-wired to 0.
9	0	Phantom Functions Enable - 81341 and 81342 does not use phantom functions. Hard-wired to 0.
8	0	Extended Tag Field Enable - 81341 and 81342 does not generate 8 bit tags. Hard-wired to 0.
7:5	000	Max_Payload_Size – This field sets maximum TLP payload size for the device. As a receiver, the device must handle TLPs as large as the set value; as transmitter, the device must not generate TLPs exceeding the set value. Defined encodings for this field are: 000b 128B max payload size 001b 256B max payload size 010b 512B max payload size 011b 1024B max payload size (Unsupported) 100b 2048B max payload size (Unsupported) 101b 4096B max payload size (Unsupported) 110b Reserved 111b Reserved Any unsupported or reserved value is treated as 128B. Note: In a multifunction configuration, the minimum programmed value from all functions is used when transmitting packets, and checking for max_payload violations.
4	1	Enable Relaxed Ordering
3	0	Unsupported Request Reporting Enable – This bit in conjunction with other bits controls the signaling of Unsupported Requests by sending Error Messages. For a multi-function device, this bit controls error reporting from the point-of-view of the respective function.

**Table 193. PCI Express Device Control Register - PE_DCTL (Sheet 2 of 2)**

Internal Bus Address Offset +0D8H 15 12 8 4 0 IOP Attributes pr rw rw rw rw ro ro ro rw rw rw rw rw rw rw PCI Attributes pr rw rw rw rw ro ro ro rw rw rw rw rw rw rw rw Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
2	0	Fatal Error Reporting Enable – This bit in conjunction with other bits control sending ERR_FATAL messages. For a multi-function device, this bit controls error reporting for each function from the point-of-view of the respective function. For a Root Port, the reporting of fatal errors is internal to the root. No external ERR_FATAL message is generated.
1	0	Non-Fatal Error Reporting Enable – This bit in conjunction with other bits controls sending ERR_NONFATAL messages. For a multi-function device, this bit controls error reporting from the point-of-view of the respective function. For a Root Port, the reporting of non-fatal errors is internal to the root. No external ERR_NONFATAL message is generated.
0	0	Correctable Error Reporting Enable – This bit in conjunction with other bits controls sending ERR_COR messages. For a multi-function device, this bit controls error reporting from the point-of-view of the respective function. For a Root Port, the reporting of correctable errors is internal to the root. No external ERR_COR message is generated.



3.16.61 PCI Express* Link Capabilities Register - PE_LCAP

This register identifies the capabilities and current operating mode of ATU, DMAs and Message Unit when operating in the PCI Express* mode.

Table 195. PCI Express* Link Capabilities Register - PE_LCAP

		31	28	24	20	16	12	8	4	0																					
IOP Attributes	[	ro	ro	ro	ro	ro	ro	pr	pr	pr	pr	pr	pr	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	ro	ro	ro	ro			
PCI Attributes	[	ro	ro	ro	ro	ro	ro	ro	pr	pr	pr	pr	pr	pr	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro			
Internal Bus Address Offset +0DCH												Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set										RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible									

Bit	Default	Description
31:24	00H	Port # - PCI Express* port number.
23:18	00H	Preserved
17:15	111b	L1 Exit Latency- Active State L1 Transition not supported
14:12	001b	L0s Exit Latency - 64ns - 128ns.
11:10	01b	Active State Link PM Support
9:4	08H	Maximum Link Width - This device supports a maximum width of x8.
3:0	1H	Maximum Link Speed - The PCI Express Link operates at 2.5 Gb/s.



3.16.62 PCI Express Link Control Register - PE_LCTL

This register controls various modes and features of ATU and Message Unit when operating in the PCI Express mode.

Table 196. PCI Express Link Control Register PE_LCTL

Internal Bus Address Offset +0E0H 15 12 8 4 0 IOP Attributes pr pr pr pr pr pr pr pr rw rw rw rw ro pr rw rw PCI Attributes pr pr pr pr pr pr pr pr rw rw rw rw ro pr rw rw Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:8	00 ₂	Preserved
7	0	Extended Synch - This bit when set forces the transmission of 4096 FTS ordered sets in the L0s state followed by a single SKP ordered set prior to entering the L0 state, and the transmission of 1024 TS1 ordered sets in the L1 state prior to entering the Recovery state. This mode provides external devices (e.g., logic analyzers) monitoring the Link time to achieve bit and Symbol lock before the Link enters the L0 or Recovery states and resumes communication.
6	0	Common Clock Configuration - When set indicates that this component and the component at the opposite end of this Link are operating with a distributed common reference clock. This bit used to report the correct L0s and L1 Exit Latencies in the PCIE_LCAP register
5	0	Retrain Link - As an end point, this bit is hard-wired to 0 As a root complex, this bit initiates Link Retraining when set. This bit is self clearing and always returns 0 when read.
4	0	Link Disable - As an endpoint, this bit is hard-wired to 0 As a root complex, this bit disables the Link when set to 1b. Writes to this bit are immediately reflected in the value read from the bit, regardless of actual Link state.
3	Endpoint: 0 Root Complex: 1	Read Completion Boundary (RCB) Control - Indicates the Root Complex's RCB. As an end point, this field is not supported and is hard-wired to 0. For Root Complex, Hard-wired to 1b indicating 128Byte RCB
2	0	Reserved
1:0	00	Active State PM Control - This field controls the level of active state PM supported on the given PCI Express Link.



3.16.63 PCI Express Link Status Register - PE_LSTS

This register controls various modes and features of ATU and Message Unit when operating in the PCI Express mode.

Table 197. PCI Express Link Status Register PE_LSTS

Internal Bus Address Offset +0E2H			Attribute Legend: RW = Read/Write RZ = Reserved Zero RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible	
IOP Attributes [15 12 8 4 0] PCI Attributes [15 12 8 4 0]				
Bit	Default	Description		
15:13	0H	Reserved		
12	1	Slot Clock Configuration - Indicates that the component uses the same physical reference clock that the platform provides on the connector. This bit must be cleared when the device uses an independent reference clock.		
11	0	Link Training - As an endpoint, this bit is hard-wired to 0 For root complex, this read-only bit indicates that Link training is in progress. Hardware clears this bit once Link training is complete.		
10	0	Link Training Error - As an endpoint, this bit is hard-wired to 0 For root complex, this read-only bit indicates that a Link training error occurred. This bit is cleared by hardware upon successful training of the Link to L0 link state.		
9:4	00H	Negotiated Link Width - Defined encodings are 01H x1 02H x2 04H x4 08H x8 12H x12 (Unsupported) 10H x16 (Unsupported) 20H x32 (Unsupported) All other encodings are reserved		
3:0	1H	Link Speed - Negotiated Link Speed. 1H indicates 2.5Gb/s Link speed. All other encodings are reserved.		



3.16.64 PCI Express Slot Capabilities Register - PE_SCAP

This register identifies PCI Express slot specific capabilities.

Table 198. PCI Express Slot Capabilities Register - PE_SCAP

312824201612840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw pr pr rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw 312824201612840 PCI Attributes rv rv rv rv ro ro ro ro ro ro ro ro pr pr ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +0E4H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:19	0000H	Physical Slot Number This field indicates the physical slot number attached to the Port. The hardware initialized value must be assigned a slot number that is globally unique within the chassis. These registers should be initialized to 0 for ports connected to devices that are integrated on the system board.
18:17	00b	Reserved
16:15	00b	Slot Power Limit Scale
14:7	00H	Slot Power Limit Value
6	0	Hot-Plug Capable
5	0	Hot-Plug Surprise
4	0	Power Indicator Present
3	0	Attention Indicator Present
2	0	MRL Sensor Present
1	0	Power Controller Present
0	0	Attention Button Present



3.16.65 PCI Express* Slot Control Register - PE_SCR

This register controls PCI Express* Slot specific parameters.

81341 and 81342 does not implement Hot-Plug support for its downstream ports when operating as a root complex. This is left as R/W for the IOP in case a software solution can be implemented using the GPIO pins.

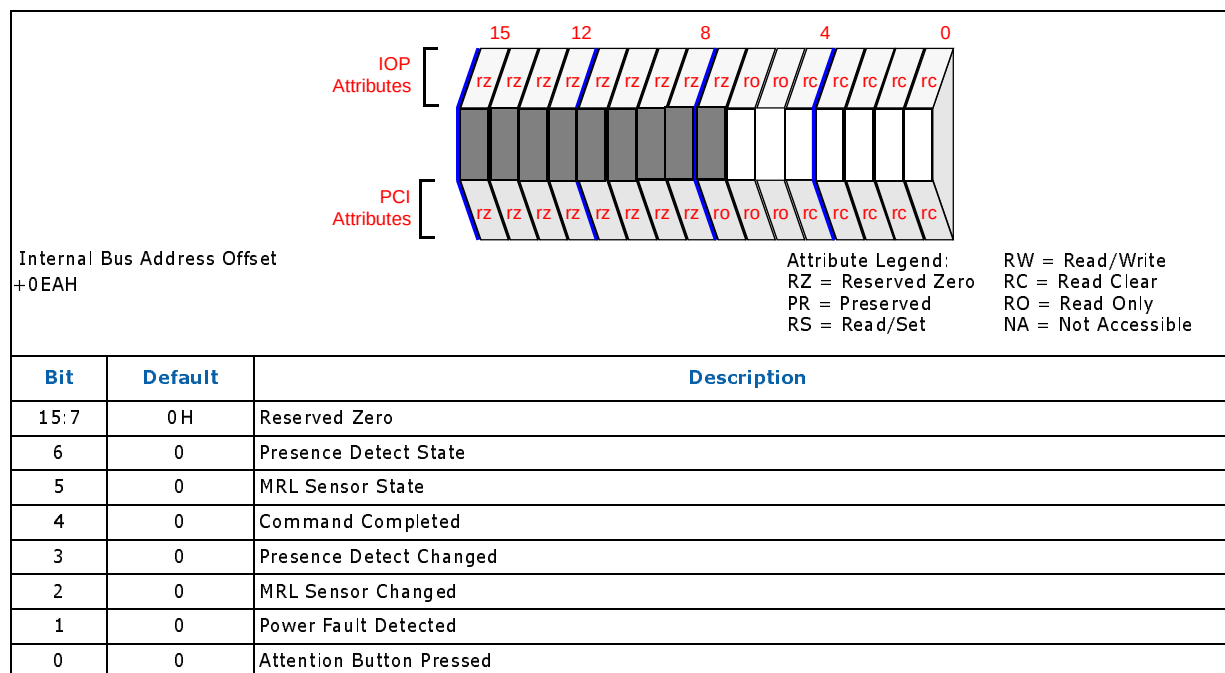
Table 199. PCI Express Slot Control Register PE_SCR

15 12 8 4 0 IOP Attributes pr pr pr pr pr rw rw rw rw rw rw rw rw rw rw PCI Attributes pr pr pr pr pr rw rw rw rw rw rw rw rw rw rw Internal Bus Address Offset +0E8H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:11	0H	Reserved
10	0	Power Controller Control
9:8	00	Power Indicator Control
7:6	00	Attention Indicator Control
5	0	Hot-Plug Interrupt Enable
4	0	Command Completed Interrupt Enable
3	0	Presence Detect Changed Enable
2	0	MRL Sensor Changed Enable
1	0	Power Fault Detected Enable
0	0	Attention Button Power Enable



This register provides information about PCI Express Slot specific parameters.

Table 200. PCI Express Slot Status Register PE_SSTS





3.16.67 PCI Express Root Control Register - PE_RCR

This register controls PCI Express Slot specific parameters.

Table 201. PCI Express Root Control Register - PE_RCR

15 12 8 4 0 IOP Attributes pr pr pr pr pr pr pr pr pr pr pr rw rw rw rw rw PCI Attributes pr pr pr pr pr pr pr pr pr pr pr rw rw rw rw rw Internal Bus Address Offset +0ECH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:4	0H	Reserved
3	0	PME Interrupt Enable This bit when set enables interrupt generation upon receipt of a PME message as reflected in the PME Status register bit. A PME Interrupt is also generated when the PME Status register bit is set when this bit is set from a cleared state.
2	0	System Error on Fatal Error Enable When set, the ATU generates the ATU_SERR interrupt when a fatal error (ERR_FATAL) message is received or a fatal error is detected by ATU. This is only valid when operating as the root complex
1	0	System Error on Non-Fatal Error Enable When set, the ATU generates the ATU_SERR interrupt when a non-fatal error (ERR_NONFATAL) message is received or a non-fatal error is detected by ATU. This is only valid when operating as the root complex
0	0	System Error on Correctable Error Enable When set, the ATU generates the ATU_SERR interrupt when a correctable error (ERR_COR) message is received or a correctable error is detected by ATU. This is only valid when operating as the root complex



3.16.68 PCI Express Root Status Register - PE_RSR

The Root Status Register provides information about PCI Express device specific parameters.

Table 202. PCI Express Root Status Register PE_RSR

		Internal Bus Address Offset +0F0H																													
		Attribute Legend: RW = Read/Write RZ = Reserved Zero RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																													
Bit	Default	Description																													
31:18	0000H	Reserved Zero - Software must write 0 to these bits																													
17	0	PME Pending: The read-only bit indicates that another PME is pending when the PME Status bit is set. When the PME Status bit is cleared by software; the PME is delivered by hardware by setting the PME Status bit again and updating the Requester ID appropriately. The ATU only supports a single PME at a time. This bit hard-wired to 0.																													
16	0	PME Status: This bit indicate that PME was asserted by the requestor ID indicated in the PME requestor ID field. Subsequent PMEs are dropped until the status register is cleared by software by writing a 1 to this bit.																													
15:0	0000H	PME Requestor ID This field indicates the PCI requestor ID of the first PME requestor.																													



3.16.69 PCI Express* Advanced Error Capability Identifier - ADVERR_CAPID

This register stores the PCI Express* extended capability ID value.

Table 203. PCI Express Advanced Error Capability Identifier - ADVERR CAPID

Internal Bus Address Offset +100H Attribute Legend: RV = Reserved, PR = Preserved, RS = Read/Set, RW = Read/Write, RC = Read Clear, RO = Read Only, NA = Not Accessible		
Bit	Default	Description
31:20	000H	Next PCI Express* Extended Capability Pointer: This is the last capability. Program with 1E0H to point to the PCI Express* Device Serial Number as the next capability, or program with 1F0H to bypass the DSN and point to Power Budgeting as the next capability.
19:16	1H	Advanced Error Capability Version Number: PCI Express* Advanced Error Reporting Extended Capability Version Number.
15:0	0001H	Advanced Error Capability ID: PCI Express* Extended Capability ID indicating Advanced Error Reporting Capability.

3.16.70 PCI Express* Uncorrectable Error Status - ERRUNC_STS

The Uncorrectable Error Status register indicates error detection status of individual uncorrectable errors on a PCI Express device. An individual error status bit that is set to "1" indicates that a particular error was detected; software may clear an error status by writing a 1 to the respective bit.

Note: All bits in this register are sticky through reset.

Table 204. PCI Express* Uncorrectable Error Status - ERRUNC STS

Internal Bus Address Offset
+104H

Attribute Legend:
 RZ = Reserved Zero
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

Bit	Default	Description
31:21	0	Reserved Zero - Software must write 0 to these bits.
20	0	Unsupported Request Error Status - As a receiver, Set whenever an unsupported request is detected. The Header is logged.
19	0	ECRC Check - As a receiver, set when ECRC check fails. The Header is logged.
18	0	Malformed TLP: As a receiver, set whenever a malformed TLP is detected. The Header is logged.
17	0	Receiver Overflow: Set when PCI Express* receive buffers overflow.
16	0	Unexpected Completion: As a receiver, set whenever a completion is received that does not match the 81341 and 81342's requestor ID or outstanding Tag. The Header is logged.
15	0	Completer Abort: As a completer, set whenever an internal agent signals a data abort. The header is logged.
14	0	Completion Timeout: As a requester, set whenever an outbound Non Posted Request does not receive a completion within 16-32ms.
13	0	Flow Control Protocol Error Status: Set whenever a flow control protocol error is detected.
12	0	Poisoned TLP Received: As a receiver, set whenever a poisoned TLP is received from PCI Express*. The header is logged.
11:5	0	Reserved Zero - Software must write 0 to these bits.
4	0	Data Link Protocol Error: Set whenever a data link protocol error is detected.
3:1	0	Reserved Zero - Software must write 0 to these bits.
0	0	Training Error - Set whenever a training error is detected.

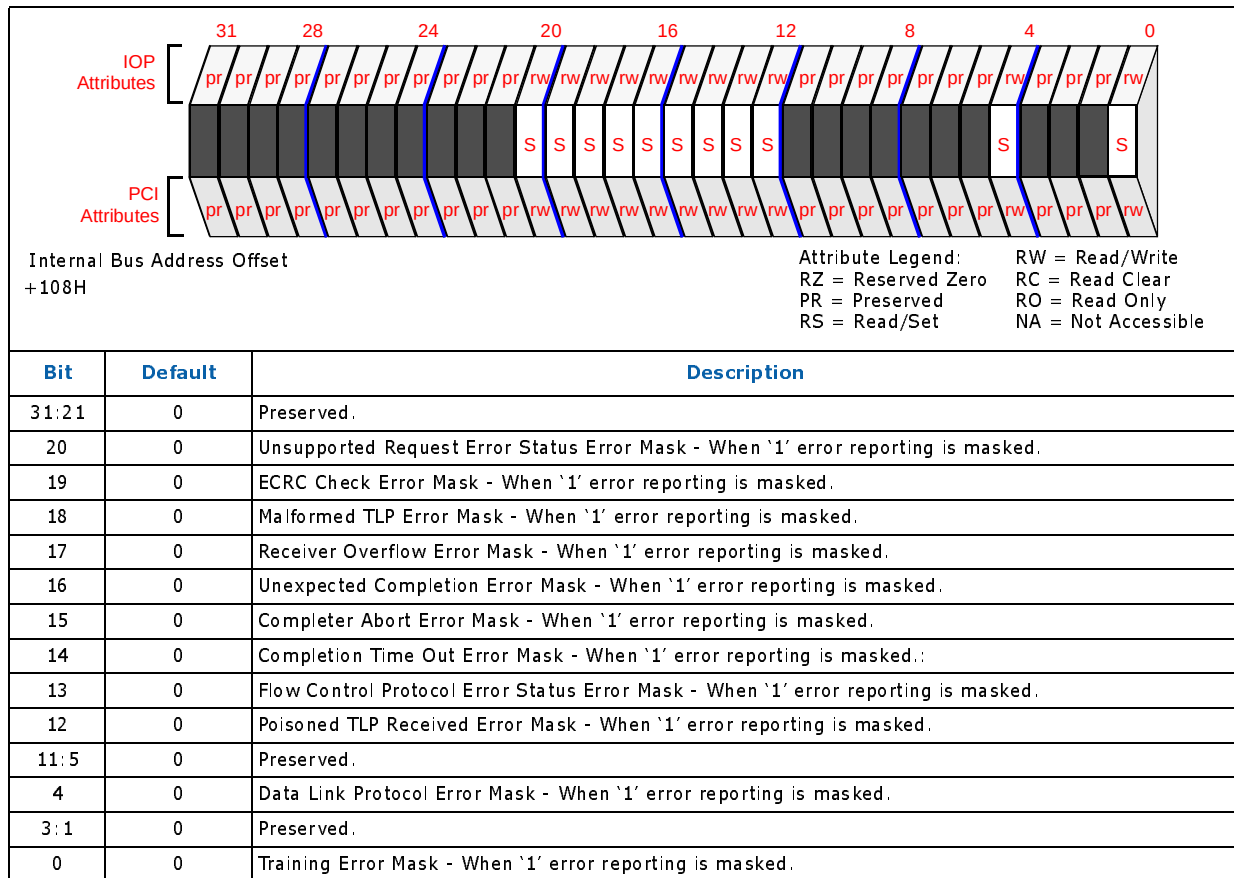


3.16.71 PCI Express* Uncorrectable Error Mask - ERRUNC_MSK

The Uncorrectable Error Mask register controls reporting of individual errors by the device to the PCI Express Root Complex via a PCI Express error message. A masked error (respective bit set to 1b in the mask register) is not logged in the Header Log register, does not update the First Error Pointer, and is not reported to the PCI Express Root Complex by an individual device.

Note: All bits in this register are sticky through reset.

Table 205. PCI Express Uncorrectable Error Mask - ERRUNC_MSK





3.16.72 PCI Express* Uncorrectable Error Severity - ERRUNC_SEV

The Uncorrectable Error Severity register controls whether an individual uncorrectable error is reported as a non-fatal or fatal error. An error is reported as fatal when the corresponding error bit in the severity register is set. When the bit is cleared, the corresponding error is considered non-fatal.

Note: All bits in this register are sticky through reset.

Table 206. PCI Express Uncorrectable Error Severity - ERRUNC_SEV

IOP Attributes PCI Attributes 312824201612840 pr pr pr pr pr pr pr pr pr pr pr pr rw rw rw rw rw rw rw rw pr		

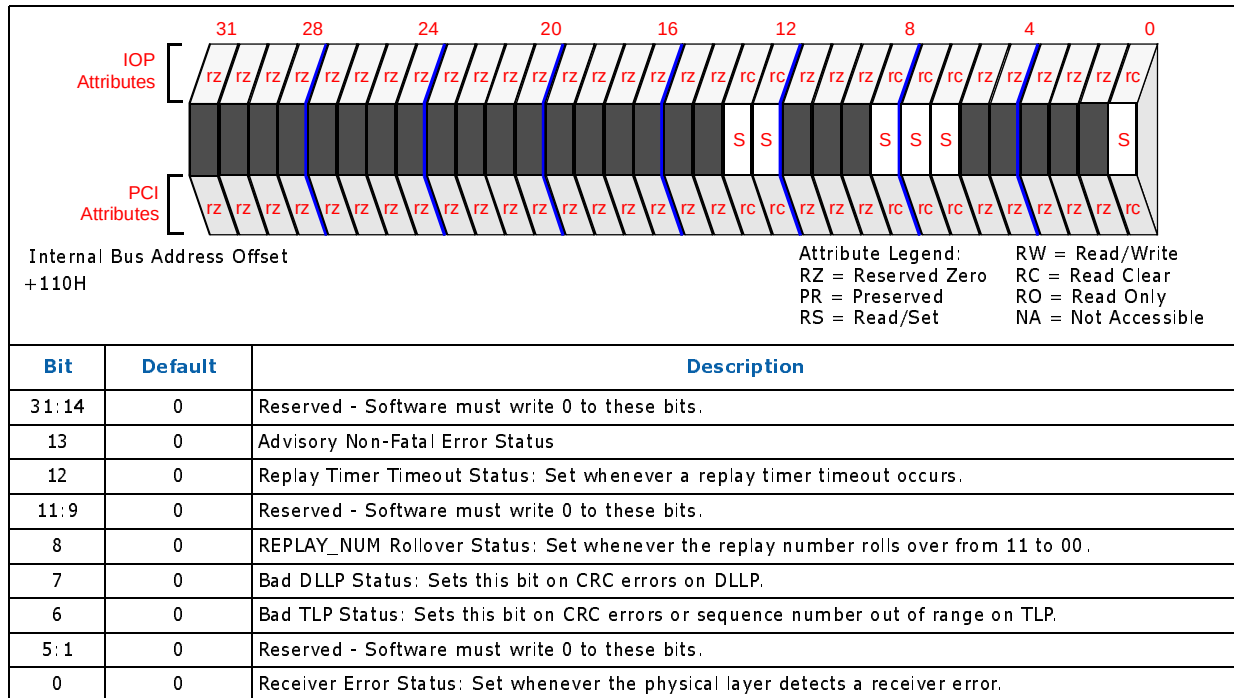


3.16.73 PCI Express* Correctable Error Status - ERRCOR_STS

The Correctable Error Status register reports error status of individual correctable error sources on a PCI Express* device. When an individual error status bit is set to "1" it indicates that a particular error occurred; software may clear an error status by writing a 1 to the respective bit

Note: All bits in this register are sticky through reset.

Table 207. PCI Express Correctable Error Status - ERRCOR_STS





3.16.74 PCI Express* Correctable Error Mask - ERRCOR_MSK

The Correctable Error Mask register controls reporting of individual correctable errors via ERR_COR message. A masked error (respective bit set in mask register) is not reported to the PCI Express Root Complex. There is a mask bit per error bit in the Correctable Error Status register.

Note: All bits in this register are sticky through reset.

Table 208. PCI Express Correctable Error Mask - ERRCOR_MSK

Internal Bus Address Offset +114H																																	
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set														RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																	
Bit	Default	Description																															
31:14	0	Preserved.																															
13	1	Advisory Non-Fatal Error Mask - this bit is set by default to enable compatibility with software that does not comprehend Role-Based Error Reporting.																															
12	0	Replay Timer Timeout Mask																															
11:9	0	Preserved.																															
8	0	REPLAY_NUM Rollover Mask																															
7	0	Bad DLLP Mask																															
6	0	Bad TLP Mask																															
5:1	0	Preserved.																															
0	0	Receiver Error Mask																															

3.16.75 Advanced Error Control and Capability Register - ADVERR_CTL

The register gives the status and control for ECRC checks and also the pointer to the first uncorrectable error that happened.

Note: All bits in this register are sticky through reset.

Table 209. Advanced Error Control and Capability Register - ADVERR_CTL

Internal Bus Address Offset +118H

Attribute Legend:
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

Bit	Default	Description
31:9	0	Reserved.
8	0	ECRC Check Enable - When set enables ECRC checking.
7	1	ECRC Check Capable - Indicates the ATU is capable of checking ECRC.
6	0	ECRC Generation Enable - When set enables ECRC generation.
5	1	ECRC Generation Capable - Indicates the ATU is capable of generating ECRC.
4:0	0 0000	The First Error Pointer - Identifies the bit position of the first error reported in the PCI Express Uncorrectable Error Status - ERRUNC_STS register. Note: This register does not update until all bits in the ERRUNC_STS register are cleared.

3.16.76 PCI Express* Advanced Error Header Log - ADVERR_LOG0

Transaction header log for PCI Express* error.

Table 210. PCI Express Advanced Error Header Log - ADVERR_LOG0

Bit	Default	Description
31:0	0	1st DWord of the Header for the PCI Express* packet in error. Once an error is logged in this register, it remains locked for further error logging until the time the software clears the status bit that cause the header log i.e. the error pointer is rearmed to log again.



3.16.77 PCI Express* Advanced Error Header Log - ADVERR_LOG1

Transaction header log for PCI Express* error.

Table 211. PCI Express Advanced Error Header Log - ADVERR_LOG1

312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +120H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible				
Bit	Default	Description		
31:0	0	2nd DWord of the Header for the PCI Express* packet in error. Once an error is logged in this register, it remains locked for further error logging until the time the software clears the status bit that cause the header log i.e. the error pointer is rearmed to log again.		

3.16.78 PCI Express* Advanced Error Header Log - ADVERR_LOG2

Transaction header log for PCI Express* error.

Table 212. PCI Express Advanced Error Header Log - ADVERR_LOG2

312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro S S S S S S S S S S S S S S S S S S S S S S S S S S S S S S PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +124H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible				
Bit	Default	Description		
31:0	0	3rd DWord of the Header for the PCI Express* packet in error. Once an error is logged in this register, it remains locked for further error logging until the time the software clears the status bit that cause the header log i.e. the error pointer is rearmed to log again.		



3.16.79 PCI Express* Advanced Error Header Log - ADVERR_LOG3

Transaction header log for PCI Express* error.

Table 213. PCI Express Advanced Error Header Log - ADVERR_LOG3

IOP Attributes PCI Attributes 312824201612840 ro																															

3.16.80 Root Error Command Register - RERR_CMD

The Root Error Command Register is used to notify the Intel XScale® processor in response to PCI Express Error Messages. This bits enable or disable the generation of the ATU Root Complex error interrupt.

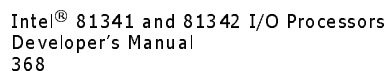
Table 214. Root Error Command Register - RERR_CMD

Internal Bus Address Offset +12CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																													
31:3	0000 0000H	Preserved																													
2	0	Fatal Error Reporting Enable When set, this bit enables the generation of the ATU Root Complex Error interrupt.																													
1	0	Non-Fatal Error Reporting Enable When set, this bit enables the generation of the ATU Root Complex Error interrupt																													
0	0	Correctable Error Reporting When set, this bit enables the generation of the ATU Root Complex Error interrupt																													



When operating as the root complex of the PCI Express domain, the Root Error Status Register reports the status of error messages received by the ATU, and of errors detected by the ATU. Each correctable and uncorrectable (FATAL/NONFATAL) error source has a first error bit and a next error bit. When an error is received by the ATU, the respective first error bit is set and the Requester ID is logged in the Error Source Identification Register. This register is updated regardless of the settings of the Root Control register and the Root Error Command registers.

Table 215. Root Error Status Register - RERR_SR





3.16.82 Error Source Identification Register - RERR_ID

The Root Error Command Register is used to notify the Intel XScale® processor in response to PCI Express Error Messages. This bits enable or disable the generation of the ATU Root Complex error interrupt.

Note: All bits in this register are sticky through reset.

Table 216. Error Source Identification Register RERR_ID

		Internal Bus Address Offset +134H																																			
		Attribute Legend: RW = Read/Write, RV = Reserved, PR = Preserved, RS = Read/Set, RC = Read Clear, RO = Read Only, NA = Not Accessible																																			
Bit	Default	Description																																			
31:16	0000H	ERR_FATAL/NONFATAL Source Identification Register This field loaded with the Requester ID indicated in the received ERR_COR message																																			
15:0	0000H	ERR_COR Source Identification Register																																			

3.16.83 Device Serial Number Capability - DSN_CAP

The Device Serial Number is a read-only 64-bit value that is unique for a given PCI Express device.

Table 217. Device Serial Number Capability - DSN_CAP

		Internal Bus Address Offset +1E0H																																			
		Attribute Legend: RW = Read/Write, RV = Reserved, PR = Preserved, RS = Read/Set, RC = Read Clear, RO = Read Only, NA = Not Accessible																																			
Bit	Default	Description																																			
31:20	1F0H	Next Capability Offset - Points to the PCI Express* Power Budgeting Capability as the next capability.																																			
19:16	1H	Capability Version - This field is the PCI-SIG defined version number.																																			
15:0	0003H	PCI Express Extended Capability ID Extended Capability ID for the Device Serial Number Capability is 0003h.																																			



The Serial Number register is a 64-bit field that contains the IEEE defined 64-bit extended unique identifier (EUI-64™). This identifier includes a 24-bit company id value assigned by IEEE registration authority and a 40-bit extension identifier assigned by the manufacturer.

Internal Bus Address Offset
+1E4H

Attribute Legend:
 RV = Reserved
 RC = Read Clear
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible

Bit	Default	Description
31:0	0H	PCI Express Device Serial Number (Lower DW) This register represents bits 31 to 0 of the EUI-64 identifier.

The Serial Number register is a 64-bit field that contains the IEEE defined 64-bit extended unique identifier (EUI-64). This identifier includes a 24-bit company id value assigned by IEEE registration authority and a 40-bit extension identifier assigned by the manufacturer.

Internal Bus Address Offset +1E8H		
Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RS = Read/Set RW = Read/Write RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:0	0H	PCI Express Device Serial Number (Upper DW) This register represents bits 63 to 32 of the EUI-64 identifier.



3.16.86 PCI Express Advisory Error Control Register - PIE_AEC

This registers enables Advisory Error functionality for devices that attempts to recover from certain errors. Firmware can attempt to recover from these non-fatal conditions zero, once, or more (finite) times. Once the retry limit has been reached the generate ERR_NONFATAL bit should be set to alert the host to problem.

Table 220. PCI Express Advisory Error Control Register PIE_AEC

Internal Bus Address Offset +1ECH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:07	0	Reserved
6	0	Advisory Error Enable for Completion Timeout - This bit determines the behavior when detecting a completion timeout. 0 = Do not treat as an Advisory Error. Send ERR_NONFATAL to root complex when error is detected 1 = Treat as Advisory Error and send ERR_COR to root complex See Section 3.9.1.8, "Completion Timeout" on page 278 for more details.
5	0	Advisory Error Enable for Poisoned Posted TLPs - This bit determines the behavior when receiving a posted transaction that has poisoned data. 0 = Do not treat as an Advisory Error. Send ERR_NONFATAL to root complex when error is detected 1 = Treat as Advisory Error and send ERR_COR to root complex See Section 3.9.1.7, "Poisoned TLP Received" on page 278 for more details.
4	0	Advisory Error Enable for Poisoned Completions - This bit determines the behavior when receiving a completion transaction that has poisoned data. 0 = Do not treat as an advisory error. Send ERR_NONFATAL to root complex when error is detected 1 = Treat as Advisory Error and send ERR_COR to root complex See Section 3.9.1.7, "Poisoned TLP Received" on page 278 for more details.
3:2	00	Reserved
1	0	Generate ERR_NONFATAL - When asserted an ERR_NONFATAL message is sent to the root complex. This bit is self-clearing and always returns zero when read.
0	0	Reserved



3.16.87 Power Budgeting Enhanced Capability Header - PWRBGT_CAPID

This register defines the power budgeting capability identifier.

Table 221. Power Budgeting Enhanced Capability Header - PWRBGT_CAPID

IOP Attributes PCI Attributes 312824201612840 ro																															

3.16.88 Power Budgeting Data Select Register - PWRBGT_DSEL

This register defines the power budgeting capability identifier.

Table 222. Power Budgeting Data Select Register - PWRBGT_DSEL

31 28 24 20 16 12 8 4 0 IOP Attributes pr																															

3.16.89 Power Budgeting Data Register - PWRBGT_DATA

This read-only register returns the DWORD of Power Budgeting Data selected by the Data Select Register. The values of one of the [Power Budgeting Information Registers\[0:23\]](#)—PWRBGT_INFO[0:23] is returned when this register is read.

Table 223. Power Budgeting Data Register - PWRBGT_DATA

Bit	Default	Description
31:21	000H	Preserved
20:18	000	Power Rail - Specifies the power rail of the operating condition being described. Defined encodings are: 000 Power (12V) 001 Power (3.3V) 010 Power (1.8V) 111 Thermal All other encodings are reserved.
17:15	000	Type - Specifies the type of operating condition being described. Defined encodings are: 000 PME Aux 001 Auxiliary 010 Idle 011 Sustained 111 Maximum All other encodings are reserved.
14:13	000	PM State - Specifies the power management state of the operating condition being described. 00 D0 01 D1 10 D2 11 D3 A device returns 11b in this field and Aux or PME Aux in the Type register to specify the D3-Cold PM state. An encoding of 11b with any other Type register value specifies the D3-Hot state.
12:10	000	PM Sub State - Specifies the power management sub state of the operating condition being described. 000 Default Sub State 001 - 111 Device Specific Sub State
9:8	00	Data Scale - Specifies the scale to apply to the Base Power Value. Defined encodings are: 00 1.0x 01 0.1x 10 0.01x 11 0.001x
7:0	00H	Base Power - Specifies in watts the base power value in the given operating condition. This value must be multiplied by the data scale to produce the actual power consumption value.



Table 224. Power Budgeting Capability Register - PWRBGT CAP

Intel® 81341 and 81342 I/O Processors
Developer's Manual
374



3.16.91 Power Budgeting Information Registers[0:23]—PWRBGT_INFO[0:23]

There are 24 power budgeting information registers that are used to report power consumption information for power states as defined in the *PCI Express Base Specification*, Revision 1.0a. These registers are reflected in the [Power Budgeting Data Register - PWRBGT_DATA](#) based on the setting of the Data Select field in the [Power Budgeting Data Select Register - PWRBGT_DSEL](#).

Data Select	PWRBGT_INFOx Register	Offset
00h	0	+200H
01h	1	+204H
02h	2	+208H
03h	3	+20CH
04h	4	+210H
05h	5	+214H
06h	6	+218H
07h	7	+21CH
08h	8	+220H
09h	9	+224H
0Ah	10	+228H
0Bh	11	+22CH

Data Select	PWRBGT_INFOx Register	Offset
0Ch	12	+230H
0Dh	13	+234H
0Eh	14	+238H
0Fh	15	+23CH
10h	16	+240H
11h	17	+244H
12h	18	+248H
13h	19	+24CH
14h	20	+250H
15h	21	+254H
16h	22	+258H
17h	23	+25CH

The lower 8 bits of the offset can be determined by shifting the Data Select value left by 2 (i.e. multiply by 4). Currently the default values are all 0, this may change once we get real power/thermal numbers for 81341 and 81342.

Table 225. Power Budgeting Information Registers[0:23]—PWRBGT_INFO[0:23]

Internal Bus Address Offset +200 through +25FH		Attribute Legend: RZ = Reserved Zero PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:21	000H	Preserved
20:0	00000H	See "Power Budgeting Data Register - PWRBGT_DATA" on page 373 for format.



3.16.92 Outbound I/O Base Address Register - OIOBAR

The OIOBAR register locates the 64 KB I/O cycle address window in the 81341 and 81342's 64 Gbyte internal address space. When A[35:16] of the internal bus address matches the value in OIOBAR, the ATU claims the transaction and forward it over to the PCI interface as an I/O cycle.

Note: In translating the internal bus address A[35:0] for the PCI bus I/O cycle, A[15:0] is forwarded over to the PCI bus unmodified while A[31:16] is replaced with the bits 31:16 from the "Outbound I/O Window Translate Value Register - OIOWTVR".

Table 226. Outbound I/O Base Address Register - OIOBAR

312824201612840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rv rv rv rv rv rv rv rv rv rv rv rv rw rw rw PCI Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rv rv rv rv rv rv rv rv rv rv rv rv rw rw rw Internal Bus Address Offset +300H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:12	0 FFDH	Outbound I/O Base Address - This value represents bits 35 to 16 of the internal bus address used to claim an outbound I/O cycle request for the ATU.
11:3	000H	Reserved
2:0	000	Outbound I/O Function Number Mapping - The Function number in this field is used as part of the Requestor ID for outbound I/O transactions.



3.16.93 Outbound I/O Window Translate Value Register - OIOWTVR

The Outbound I/O Window Translate Value Register (OIOWTVR) contains the PCI I/O address used to convert the internal bus access to a PCI address. This address is driven on the PCI Express Link as a result of the outbound ATU address translation. See [Section 3.3.2.1, “Outbound Address Translation - Internal Bus Transactions”](#) on [page 250](#) for details on outbound address translation.

The I/O window is from 81341 and 81342 internal bus is set via the “Outbound I/O Base Address Register - OIOBAR” with the a fixed length of 64 Kbytes.

Table 227. Outbound I/O Window Translate Value Register - OIOWTVR

Bit	Default	Description
31:16	0000H	Outbound I/O Window Translate Value - Used to convert internal bus addresses to PCI addresses.
15:01	0000H	Reserved
00	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all PCI read/write transactions through OIOBAR. When clear, no swap is performed. Refer to Section 3.4, “Big Endian Byte Swapping” on page 259 for more details.



3.16.94 Outbound Upper Memory Window Base Address Register 0 - OUMBAR0

The OUMBAR0 register locates Outbound Memory Window 0 in a 4 Gbyte Memory section in the 81341 and 81342's 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR[3:0], the ATU claims the transaction and forward it over to the PCI Express interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI Express Link unmodified. The ATU constructs a 64 bit PCI address in conjunction with the "Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0" on page 379.

Table 228. Outbound Upper Memory Window Base Address Register 0 - OUMBAR0

		Internal Bus Address Offset +308H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
Bit	Default	Description																											
31	1 ₂	Outbound Window 0 Enable - When set, the ATU Outbound Memory Window 0 claims internal bus transactions when A[35:32] matches OUMBAR0[3:0].																											
30:28	000 ₂	Outbound Window 0 Function Number Mapping - Errors with Outbound Memory Window 0 transactions (PCI or Internal Bus) is associated with the function number contained in this field.																											
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 3.4, "Big Endian Byte Swapping" on page 259 for more details.																											
26:04	000 0000H	Reserved																											
3:0	1H	Outbound Upper Memory Window Base Address 0 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUX has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.																											



3.16.95 Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0

The Outbound Upper 32-bit Memory Window Translate Value Register 0 (OUMWTVR0) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a 3DW header is generated on the PCI Express Link.

Table 229. Outbound Upper 32-bit Memory Window Translate Value Register 0-OUMWTVR0

		Internal Bus Address Offset +30CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																															
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).																															



3.16.97 Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1

The Outbound Upper 32-bit Memory Window Translate Value Register 1 (OUMWTVR1) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a 3DW header is generated on the PCI Express Link.

Table 231. Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1

		Internal Bus Address Offset +314H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																															
Bit	Default	Description																															
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).																															

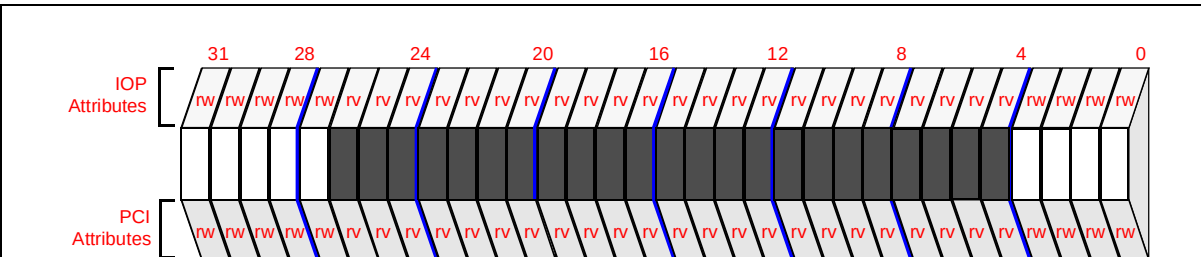


3.16.98 Outbound Upper Memory Window Base Address Register 2 - OUMBAR2

The OUMBAR2 register locates Outbound Memory Window 2 in a 4 Gbyte Memory section in the 81341 and 81342's 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR2[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the "Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2" on page 383.

Table 232. Outbound Upper Memory Window Base Address Register 2- OUMBAR2

		
Internal Bus Address Offset +318H		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	Outbound Window 2 Enable - When set, the ATU Outbound Memory Window 2 claims internal bus transactions when A[35:32] matches OUMBAR2[3:0].
30:28	000 ₂	Outbound Window 2 Function Number Mapping - Errors with Outbound Memory Window 2 transactions (PCI or Internal Bus) are associated with the function number contained in this field.
27	0	Big Endian Byte Swap enable - When set the ATU performs a byte swap on all read/write transactions through this BAR. When clear, no swap is performed. Refer to Section 3.4, "Big Endian Byte Swapping" on page 259 for more details.
26:04	000 0000H	Reserved
3:0	3H	Outbound Upper Memory Window Base Address 2 - This value represents bits 35 to 32 of the internal bus address used to claim an outbound memory request. Note: The ATUX has the same default value. When both ATUs exist and the outbound windows of the both ATUs are enabled, the bit field must be reprogrammed to avoid a conflict.



3.16.99 Outbound Upper 32-bit Memory Window Translate Value Register 2 - OUMWTVR2

The Outbound Upper 32-bit Memory Window Translate Value Register 2 (OUMWTVR2) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a 3DW header is generated on the PCI bus.

Table 233. Outbound Upper 32-bit Memory Window Translate Value Register 2- OUMWTVR2

Bit	Default	Description																											
31:00	0000 0000 H	These bits define the upper 32-bits of address.																											



The OUMBAR3 register locates Outbound Memory Window 3 in a 4 Gbyte Memory section in the 81341 and 81342's 64 Gbyte internal address space. When A[35:32] of the internal bus address matches the value in OUMBAR3[3:0], the ATU claims the transaction and forward it over to the PCI interface.

Note: In translating the internal bus address A[35:0], A[31:0] is forwarded over to the PCI bus unmodified. The ATU constructs a 64 bit PCI address in conjunction with the "Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3" on page 385.

Table 234. Outbound Upper Memory Window Base Address Register 3 - OUMBAR3

[illegible]



3.16.101 Outbound Upper 32-bit Memory Window Translate Value Register 3 - OUMWTVR3

The Outbound Upper 32-bit Memory Window Translate Value Register 3 (OUMWTVR3) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a 3DW header is generated on the PCI bus.

Table 235. Outbound Upper 32-bit Memory Window Translate Value Register 3- OUMWTVR3

		312824201612840 IOP Attributes PCI Attributes																																
Internal Bus Address Offset +324H		Attribute Legend:		RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set		RC = Read Clear RO = Read Only NA = Not Accessible																												
Bit	Default	Description																																
31:00	0000 0000H	These bits define the upper 32-bits of address.																																



3.16.102 Outbound Configuration Cycle Address Register - OCCAR

The Outbound Configuration Cycle Address Register is used to hold bytes 8-11 of the configuration transaction header. The Intel XScale® processor writes the bus, device, function, and register number which then enables the outbound configuration read or write. The Intel XScale® processor then performs a read or write to the Outbound Configuration Cycle Data Register to initiate the configuration transaction on the PCI Express Link.

Table 236. Outbound Configuration Cycle Address Register - OCCAR

312824201612840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rv rv rv rv rw rw rw rw rw rw rw rw rv rw PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset +32CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:24	00H	Bus Number
23:19	00H	Device Number
18:16	000b	Function Number
15:12	0H	Reserved
11:8	0H	Extended Register Number
7:2	00H	Register Number
1	0H	Reserved
0	0	Configuration Type 0 = Generate Type 0 Configuration TLP 1 = Generate Type 1 Configuration TLP



3.16.103 Outbound Configuration Cycle Data Register - OCCDR

The Outbound Configuration Cycle Data Register is used to initiate a configuration read or write transaction on the PCI Express Link. The register is logical rather than physical meaning that it is an address not a register. The Intel XScale® processor reads or writes the data registers memory-mapped address to initiate the configuration transaction on the PCI Express Link with the address found in the OCCAR. For a configuration write, the data is latched from the internal bus and forwarded directly to the ONPQ. For a read, the data is returned directly from the ICPLDQ to the Intel XScale® processor and is never actually entered into the data register (which does not physically exist).

The OCCDR is only visible from 81341 and 81342 internal bus address space and appears as a reserved value within the ATU configuration space.

Note: This register does not physically exist and reads from the PCI domain returns '0'.

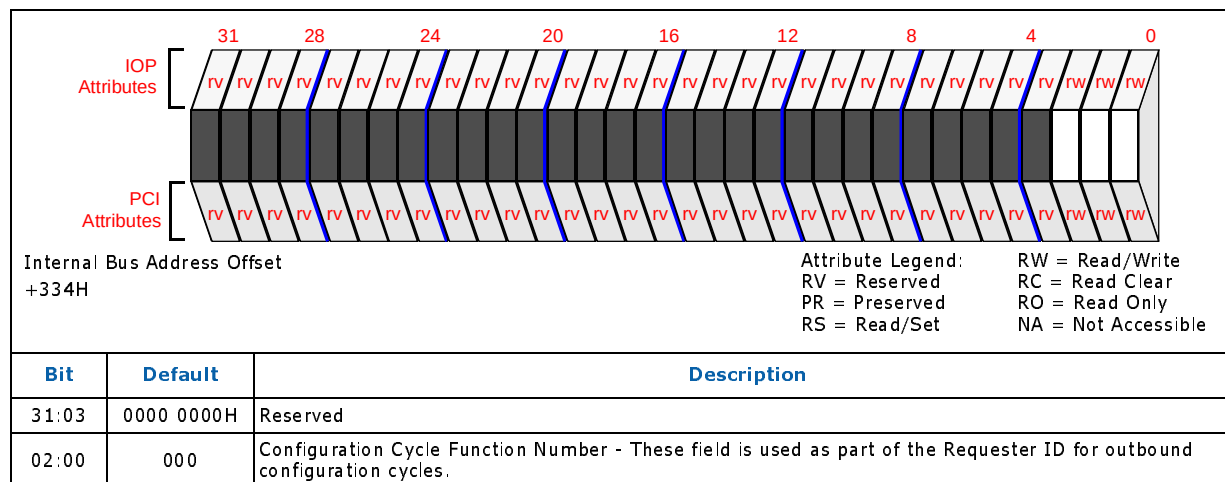
Table 237. Outbound Configuration Cycle Data Register - OCCDR

Bit	Default	Description																											
31:00	0000 0000H	Configuration Cycle Data - These bits define the data used during an outbound configuration read or write transaction.																											



This registers contains the function number that is used as part of the Requester ID for all outbound configuration requests. This field is also used to determine where errors get logged.

Table 238. Outbound Configuration Cycle Function Number - OCCFN





3.16.105 Inbound Vendor Message Header Register 0 - IVMHR0

The Inbound Vendor Message Header Registers capture the header for a vendor defined message received on the PCI Express interface. Once the inbound message has been processed, the Inbound Vendor Message Received bit is set in the [ATU Interrupt Status Register - ATUISR](#). Subsequent inbound vendor messages are held in the inbound posted queues until the status bit is cleared or the mask bit is set in the [ATU Interrupt Mask Register - ATUIMR](#). When the mask bit is set, then Vendor_Defined Type 0 messages are treated as unsupported requests and Vendor_Defined Type 1 messages are silently discarded.

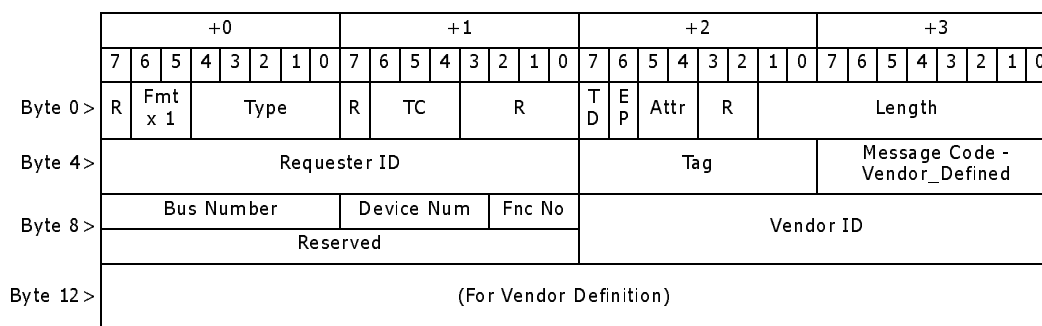
Vendor_Defined message format is shown below in [Figure 40](#)

Table 239. Inbound Vendor Defined Message Header Register0 - IVMHR0

Internal Bus Address Offset +340H																													
																Attribute Legend:													
																RW = Read/Write													
																RV = Reserved													
																PR = Preserved													
																RS = Read/Set													
																RC = Read Clear													
																RO = Read Only													
																NA = Not Accessible													

Bit	Default	Description
31:24	00H	Header Byte 0
23:16	00H	Header Byte 1
15:8	00H	Header Byte 2
7:0	00H	Header Byte 3

Figure 40. PCI Express* Vendor_Defined Message Header





The Inbound Vendor Message Header Registers capture the header for a vendor defined message received on the PCI Express* interface. Once the inbound message has been processed, the Inbound Vendor Message Received bit is set in the [ATU Interrupt Status Register - ATUISR](#). Subsequent inbound vendor messages are held in the inbound posted queues until the status bit is cleared or the mask bit is set in the [ATU Interrupt Mask Register - ATUIMR](#). When the mask bit is set, then Vendor_Defined Type 0 messages are treated as unsupported requests and Vendor_Defined Type 1 messages are silently discarded.

Internal Bus Address Offset
+344H

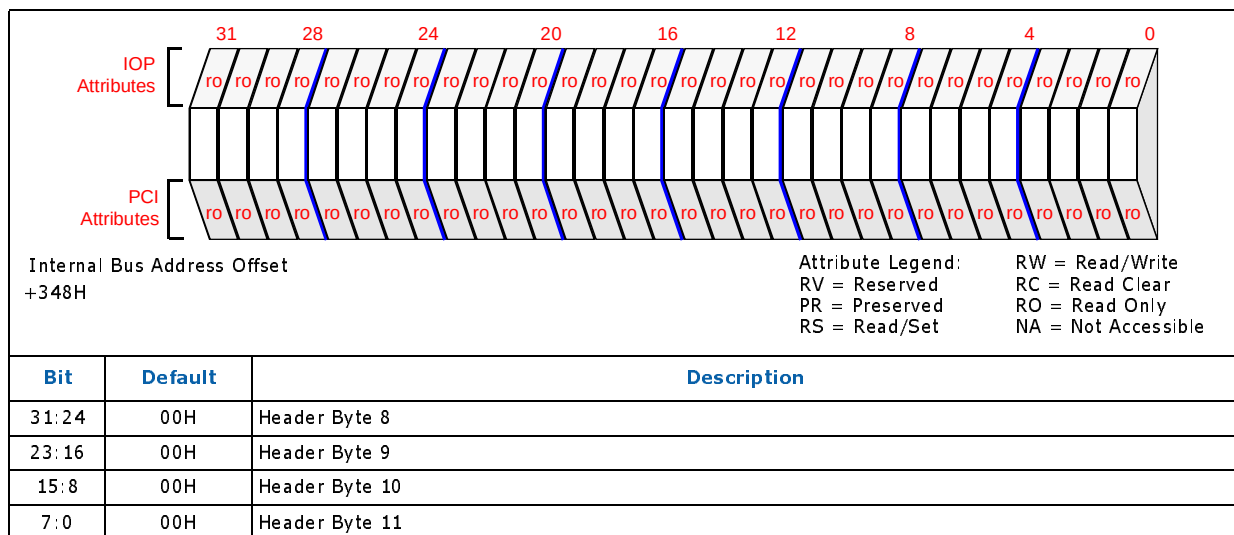
Attribute Legend:
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 PR = Preserved
 RS = Read/Set
 NA = Not Accessible



3.16.107 Inbound Vendor Message Header Register 2 - IVMHR2

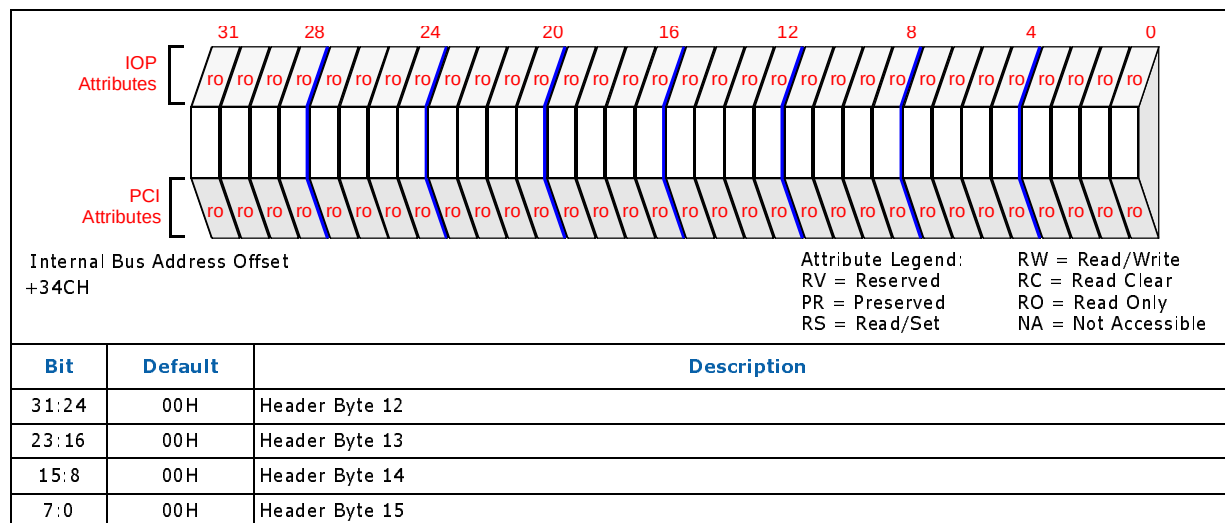
The Inbound Vendor Message Header Registers capture the header for a vendor defined message received on the PCI Express interface. Once the inbound message has been processed, the Inbound Vendor Message Received bit is set in the [ATU Interrupt Status Register - ATUISR](#). Subsequent inbound vendor messages are held in the inbound posted queues until the status bit is cleared or the mask bit is set in the [ATU Interrupt Mask Register - ATUIMR](#). When the mask bit is set, then Vendor_Defined Type 0 messages are treated as unsupported requests and Vendor_Defined Type 1 messages are silently discarded.

Table 241. Inbound Vendor Defined Message Header Register 2 - IVMHR2

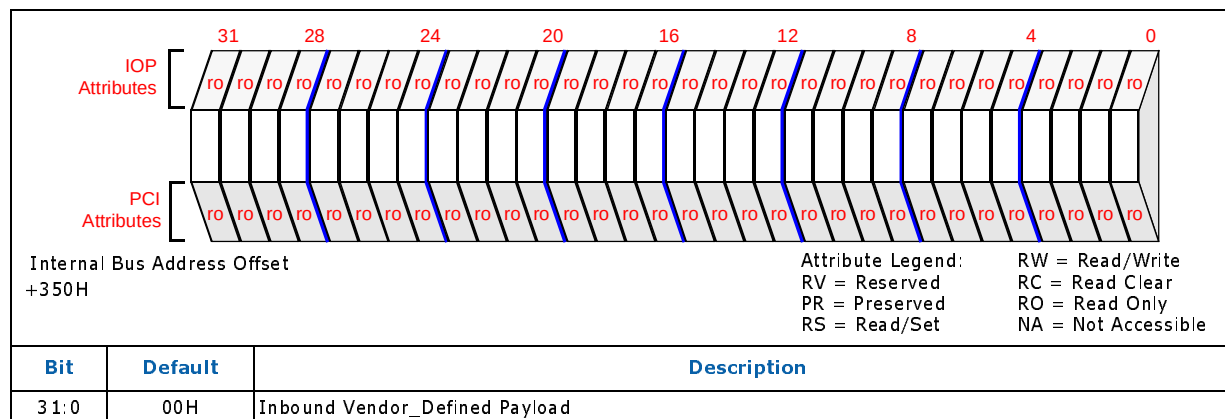




Inbound Vendor Message Header Registers capture the header for a vendor defined message received on the PCI Express interface. Once the inbound message is processed, the Inbound Vendor Message Received bit is set in the [ATU Interrupt Status Register - ATUISR](#). Subsequent inbound vendor messages are held in inbound posted queues until the status bit is cleared or mask bit is set in the [ATU Interrupt Mask Register - ATUIMR](#). When mask bit is set, Vendor_Defined Type 0 messages are treated as unsupported requests and Vendor_Defined Type 1 messages are silently discarded.



Inbound Vendor Message Payload Registers capture the payload for a vendor defined message received on the PCI Express interface. Once the inbound message has been processed, the Inbound Vendor Message Received bit is set in the [ATU Interrupt Status Register - ATUISR](#). Subsequent inbound vendor messages are held in inbound posted queues until the status bit is cleared or mask bit is set in the [ATU Interrupt Mask Register - ATUIMR](#). When the mask bit is set, Vendor_Defined Type 0 messages are treated as unsupported requests and Vendor_Defined Type 1 messages are silently discarded.





3.16.110 Outbound Vendor Message Header Register 0 - OVMHR0

The Outbound Vendor Message Header Registers allow software to create a header that is used for a Vendor_Defined Message TLP. The OVMHR0-3 registers must be programmed prior to writing the [Outbound Vendor Defined Message Payload Register - OVMPR](#). A write to the OVMPR initiates the Vendor_Defined Message TLP. When the payload length is 0, a write to the OVMPR is still required to initiate the TLP but the data written is ignored.

Vendor_Defined message format is shown in [Figure 40](#)

Table 244. Outbound Vendor Defined Message Header Register0 - OVMHR0

		<																															



3.16.111 Outbound Vendor Message Header Register 1 - OVMHR1

The Outbound Vendor Message Header Registers allow software to create a header that is used for a Vendor_Defined Message TLP. The OVMHR0-3 registers must be programmed prior to writing the [Outbound Vendor Defined Message Payload Register - OVMPR](#). A write to the OVMPR initiates the Vendor_Defined Message TLP. When the payload length is 0, a write to the OVMPR is still required to initiate the TLP but the data written is ignored.

Vendor_Defined message format is shown in [Figure 40](#)

Table 245. Outbound Vendor Defined Message Header Register 1 - OVMHR1

		Internal Bus Address Offset +364H																																			
		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:24	00H	Requester ID - Bus number. This value is copied from the Bus number field in the PCISR.																																			
23:19	0_0000	Requester ID - Device Number. This value is copied from the Device number field in the PCISR																																			
18:16	000	Requester ID - Function Number. This value is programmable.																																			
15:8	00H	Tag - The tag field is undefined for posted requests. Hardcoded to 0.																																			
7:0	0111_1110	Message Code[7:0] - Indicates the type of message being generated. Valid encodings are: 0111_1110 Vendor_Defined Type 0 0111_1111 Vendor_Defined Type 1 Other codes may result in malformed packets.																																			



3.16.112 Outbound Vendor Message Header Register 2 - OVMHR2

The Outbound Vendor Message Header Registers allow software to create a header that is used for a Vendor_Defined Message TLP. The OVMHR0-3 registers must be programmed prior to writing the [Outbound Vendor Defined Message Payload Register - OVMPR](#). A write to the OVMPR initiates the Vendor_Defined Message TLP. When the payload length is 0, a write to the OVMPR is still required to initiate the TLP but the data written is ignored. Vendor_Defined message format is shown in [Figure 40](#)

Table 246. Outbound Vendor Defined Message Header Register 2 - OVMHR2

		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
Bit	Default	Description																											
31:16	00H	Destination ID - When Type[2:0] indicates Route by ID, this field is Destination ID. Otherwise field is reserved.																											
16:0	00H	Vendor ID																											

3.16.113 Outbound Vendor Message Header Register 3 - OVMHR3

The Outbound Vendor Message Header Registers allow software to create a header that is used for a Vendor_Defined Message TLP. The OVMHR0-3 registers must be programmed prior to writing the [Outbound Vendor Defined Message Payload Register - OVMPR](#). A write to the OVMPR initiates the Vendor_Defined Message TLP. When the payload length is 0, a write to the OVMPR is still required to initiate the TLP but the data written is ignored. Vendor_Defined message format is shown in [Figure 40](#)

Table 247. Outbound Vendor Defined Message Header Register 3 - OVMHR3

		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
Bit	Default	Description																											
31:0	00H	Available for Vendor Definition.																											



The Outbound Vendor Message Payload Register contains the payload data that is used for the Vendor_Defined Message TLP. A write to this register initiates the Vendor_Defined Message on the PCI Express Interface. When a zero length payload is desired, then a write to this register is required to initiate the transaction, but the value written is ignored.

The Vendor Defined message format is shown in [Figure 40](#)

This address was chosen to put the data on lane 0 of the 81341 and 81342 internal bus.

Internal Bus Address Offset +370H Attribute Legend: - RC = Read Clear - RV = Reserved - PR = Preserved - RS = Read/Set - WO = Write Only - RW = Read/Write - RO = Read Only - NA = Not Accessible		
Bit	Default	Description
31:0	00H	Outbound Vendor_Defined Payload Reads of this register always return 0.



3.16.115 PCI Interface Error Control and Status Register - PIE_CSR

This register indicates whether or not the ATU has detected and logged a PCI interface error. The register is also used to enable the logging of additional errors. For more details, see [Section 3.9, “ATU Error Conditions” on page 274](#).

Table 249. PCI Interface Error Control and Status Register - PIE_CSR

The diagram illustrates the structure of the PCI Error Status Register (PIE_STS), a 32-bit register. The bits are grouped into four 8-bit sections, each with a specific attribute:

- Bits 31:28:** Initiator ID. Attributes: ro (Read Only).
- Bits 27:24:** General Device Error (GDE). Attributes: rv (Reserved).
- Bits 23:16:** PCI Function Number. Attributes: rv (Reserved).
- Bits 15:8:** PCI Interface Error First Error Pointer. Attributes: rv (Reserved).
- Bits 7:4:** Reserved. Attributes: rv (Reserved).
- Bits 3:0:** Reserved. Attributes: rv (Reserved).

Attribute Legend:

- ro = Read Only
- rv = Reserved
- rw = Read/Write
- rc = Read Clear
- pr = Preserved
- rs = Read/Set
- na = Not Accessible



3.16.116 PCI Interface Error Status - PIE_STS

The PCI Interface Error Status register reports error status of individual uncorrectable error sources. An individual error status bit that is set to “1” indicates that a particular error occurred; software may clear an error status by writing a 1 to the respective bit.

Note: The status bits are set even when the reporting of the error is masked in the PIE_MSK register.

Table 250. PCI Interface Error Status - PIE_STS

31 28 24 20 16 12 8 4 0 IOP Attributes rc rc rc rc rz rz rz rz rz rz rz rc rc rc rc rc rc rc rz rz rz rz rz rz rz rz rc rz rz rz rc PCI Attributes rc rc rc rc rz rz rz rz rz rz rz rc rc rc rc rc rc rc rz rz rz rz rz rz rz rc rz rz rz rc Internal Bus Address Offset +384H Attribute Legend: RW = Read/Write RZ = Reserved Zero PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31	0	Received Completion with Unsupported Request Status
30	0	Received Completion with Completer Abort Status
29	0	Poisoned TLP Transmitted Status
28	0	Transmit: Header Parity Error detected
27:21	0	Reserved Zero - Software must write 0 to these bits.
20	0	Unsupported Request Error Status - As a receiver, Set whenever an unsupported request is detected. The Header is logged.
19	0	ECRC Check - As a receiver, set when ECRC check fails. The Header is logged.
18	0	Malformed TLP: As a receiver, set whenever a malformed TLP is detected. The Header is logged.
17	0	Receiver Overflow: Set when PCI Express* receive buffers overflow.
16	0	Unexpected Completion: As a receiver, set whenever a completion is received that does not match the 81341 and 81342's requestor ID or outstanding Tag. The Header is logged.
15	0	Completer Abort: As a completer, set whenever an internal agent signals a data abort. The header is logged.
14	0	Completion Timeout: As a requester, set whenever an outbound Non Posted Request does not receive a completion within 16-32ms.
13	0	Flow Control Protocol Error Status: Set whenever a flow control protocol error is detected.
12	0	Poisoned TLP Received: As a receiver, set whenever a poisoned TLP is received from PCI Express*. The header is logged.
11:5	0	Reserved Zero - Software must write 0 to these bits.
4	0	Data Link Protocol Error: Set whenever a data link protocol error is detected.
3:1	0	Reserved Zero - Software must write 0 to these bits.
0	0	Training Error - Set whenever a training error is detected.



3.16.117 PCI Interface Error Mask - PIE_MSK

The PCI Interface Error Mask register controls reporting of individual errors by the device to the Intel XScale® processor core via an interrupt (ATUISR bit 10). A masked error (respective bit set to 1b in the mask register) is not logged in the Header Log register, does not update the First Error Pointer, and generate an interrupt to the core.

Table 251. PCI Interface Error Mask - PIE_MSK

31 28 24 20 16 12 8 4 0 IOP Attributes PCI Attributes rw rw rw rw rv pr pr pr pr pr pr rw rw rw rw rw rw rw pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr rw rw rw rw rv pr pr pr pr pr pr rw rw rw rw rw rw rw pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr pr		Internal Bus Address Offset +388H	Attribute Legend: RZ = Reserved Zero PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description	
31	0	Received Completion with Unsupported Request Mask	
30	0	Received Completion with Completer Abort Mask	
29	0	Poisoned TLP Transmitted Mask	
28	0	Transmit: Header Parity Error Mask	
27:21	0	Reserved Zero - Software must write 0 to these bits.	
20	0	Unsupported Request Error Status Error Mask - When '1' error reporting is masked.	
19	0	ECRC Check Error Mask - When '1' error reporting is masked.	
18	0	Malformed TLP Error Mask - When '1' error reporting is masked.	
17	0	Receiver Overflow Error Mask - When '1' error reporting is masked.	
16	0	Unexpected Completion Error Mask - When '1' error reporting is masked.	
15	0	Completer Abort Error Mask - When '1' error reporting is masked.	
14	0	Completion Time Out Error Mask - When '1' error reporting is masked.:	
13	0	Flow Control Protocol Error Status Error Mask - When '1' error reporting is masked.	
12	0	Poisoned TLP Received Error Mask - When '1' error reporting is masked.	
11:5	0	Preserved.	
4	0	Data Link Protocol Error Mask - When '1' error reporting is masked.	
3:1	0	Preserved.	
0	0	Training Error Mask - When '1' error reporting is masked.	



3.16.118 PCI Interface Error Header Log - PIE_LOG0

Transaction header log for PCI interface errors.

Table 252. PCI Interface Error Header Log - PIE_LOG0

		Internal Bus Address Offset +38CH																																			
		Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:0	0	1st DWord of the Header for the PCI Express* packet in error. Once an error is logged in this register, it remains locked for further error logging until the time the software clears the status bit that cause the header log i.e. the error pointer is rearmed to log again.																																			

3.16.119 PCI Interface Error Header Log 1 - PIE_LOG1

Transaction header log for PCI interface errors.

Table 253. PCI Interface Error Header Log 1 - PIE_LOG1

		Internal Bus Address Offset +390H																																			
		Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:0	0	2nd DWord of the Header for the PCI Express* packet in error. Once an error is logged in this register, it remains locked for further error logging until the time the software clears the status bit that cause the header log i.e. the error pointer is rearmed to log again.																																			



3.16.120 PCI Interface Error Header Log 2 - PIE_LOG2

Transaction header log for PCI interface errors.

Table 254. PCI Interface Error Header Log 2 - PIE_LOG2

		IOP Attributes PCI Attributes 312824201612840 ro																																			

3.16.121 PCI Interface Error Header Log - PIE_LOG3

Transaction header log for PCI interface errors.

Table 255. PCI Interface Error Header Log - PIE_LOG3

		<																																			



3.16.122 PCI Interface Error Descriptor Log

Descriptor log for transaction errors.

Table 256. PCI Interface Error Descriptor Log - PIE_DLOG

Internal Bus Address Offset +39CH		
Attribute Legend: RV = Reserved RW = Read/Write PR = Preserved RC = Read Clear RS = Read/Set RO = Read Only NA = Not Accessible		
Bit	Default	Description
31	0	Reserved
30:29	00	DMA Channel Number
28:0	0H	ADMA Descriptor Address The DMA type can be determined by the Initiator ID in the "PCI Interface Error Control and Status Register - PIE_CSR"

Note: In 81341 and 81342, bits[30:0] of this register is copied directly from the USER_ATTR[65:35] sideband bus.

3.16.123 ATU Reset Control Register - ATURCR

This register controls the link down reset bypass strap (LK_DN_RST_BYPASS#).

Note: Some bits in this register are sticky through reset.

Table 257. ATU Reset Control Register - ATURCR

Internal Bus Address Offset +3B0H		
Attribute Legend: RV = Reserved RW = Read/Write PR = Preserved RC = Read Clear RS = Read/Set RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:01	0H	Reserved
00	0H	Link Down Reset Bypass override - When this bit is set, the Link Down Reset Bypass strap is ignored and treated as when it is deasserted.



4.0 Messaging Unit

This chapter describes the Messaging Unit (MU) of the Intel® 81341 and 81342 I/O Processors (81341 and 81342). The MU is located at the front-end of the DDR SDRAM Memory Controller.

4.1 Overview

The Messaging Unit (MU) provides a mechanism for data to be transferred between the PCI bus and the Intel XScale® processor and notifying the respective system of the arrival of new data through an interrupt. The MU can be used to send and receive messages.

The MU is located on the south internal bus of the 81341 and 81342. The MU is located on the south internal bus interface of the DDR SDRAM Memory Controller Unit. Refer to [Figure 78, "Memory Controller Block Diagram" on page 565](#) of the DDR SDRAM Memory Controller Unit chapter. External PCI agents access the MU via the ATU. For example, the ATU initiates transactions on the internal bus to the MU on behalf of the external PCI agents.

The MU has four distinct messaging mechanisms. Each allows a host processor or external PCI agent and the 81341 and 81342 to communicate through message passing and interrupt generation. The four mechanisms are:

- **Message Registers** — allow the 81341 and 81342 and external PCI agents to communicate by passing messages in one of four 32-bit Message Registers. In this context, a message is any 32-bit data value. Message registers combine aspects of mailbox registers and doorbell registers. Writes to the message registers may optionally cause interrupts.
- **Doorbell Registers** — allow the 81341 and 81342 to assert the PCI interrupt signals and allow external PCI agents to generate an interrupt to the Intel XScale® processor.
- **Circular Queues** — support a message passing scheme that uses four circular queues.
- **Index Registers** — support a message passing scheme that uses a portion of the 81341 and 81342 local DDR memory to implement a large set of message registers.

Each of the above are available to the system designer at the same time. No special mode selection is needed.

4.2 Theory of Operation

The MU has four independent messaging mechanisms. There are four Message Registers that are similar to a combination of mailbox and doorbell registers. Each holds a 32-bit value and generates an interrupt when written. The two Doorbell Registers support software interrupts. When a bit is set in a Doorbell Register, an interrupt is generated.

The Circular Queues support a message passing scheme that uses 4 circular queues. The 4 circular queues are implemented in 81341 and 81342 local DDR SDRAM memory. Two queues are used for inbound messages and two are used for outbound messages. Interrupts may be generated when the queue is written.

The Index Registers use a portion of the 81341 and 81342 local DDR memory to implement a large set of message registers. When one of the Index Registers is written, an interrupt is generated and the address of the register written is captured.

Interrupt status for all interrupts is recorded in the Inbound Interrupt Status Register and the Outbound Interrupt Status Register. Each interrupt generated by the Messaging Unit can be masked.

Because of read side effects, multi-word burst transactions are not supported by the Messaging Unit. The Messaging Unit must be mapped in a non-prefetchable PCI address space to avoid read side effects. Multi-word read or write transactions made to the Messaging Unit registers causes the MU to generate an address error on the internal bus of the 81341 and 81342. Multi-word transactions made by an external PCI agent results in an error being sent to the external PCI agent. Refer to the ATU chapters for more details on how the ATUs respond to an internal bus error. As explained above the Index Registers are implemented using the 81341 and 81342 local DDR SDRAM memory. Therefore, the Index Registers do not have the same multi-word access limitations as the MU registers. Refer to the [Section 4.6, “Index Registers”](#) on [page 419](#) for more details.

All registers needed to configure and control the Messaging Unit are memory-mapped registers.

The MU is accessed by an external PCI agent via the ATU. The MU can be mapped in any 4 Kbytes of the inbound translation window in the Address Translation Unit (ATU). The MU provides the Base Address Registers ([Table 287, “MU Base Address Register - MUBAR”](#) and [“MU Upper Base Address Register - MUUBAR”](#)) which allow the MU to be relocated within the ATU translated window. This PCI address window is used for PCI transactions that access the 81341 and 81342 local memory. The PCI address of the inbound translation window is contained in the Inbound ATU Base Address Register. See [Chapter 2.0, “Address Translation Unit \(PCI-X\)”](#) or [Chapter 3.0, “Address Translation Unit \(PCI Express\)”](#) for more details on inbound ATU addressing and the ATU.

Note that since the MU is located on the internal bus of the 81341 and 81342, any PCI transaction that is targeted for the MU is first claimed by the ATU and then the ATU issues the transaction on the internal bus of the 81341 and 81342.



Figure 41. PCI Memory Map

Offsets are relative to the MU Base Address Registers (MUBAR/MUUBAR)			
Offset			
0000H	reserved		
0004H	reserved		
0008H	reserved		
000CH	reserved		
0010H	Inbound Message Register 0	4 Message Registers	
0014H	Inbound Message Register 1		
0018H	Outbound Message Register 0		
001CH	Outbound Message Register 1		
0020H	Inbound Doorbell Register	2 Doorbell Registers and 4 Interrupt Registers	
0024H	Inbound Interrupt Status Register		
0028H	Inbound Interrupt Mask Register		
002CH	Outbound Doorbell Register		
0030H	Outbound Interrupt Status Register		
0034H	Outbound Interrupt Mask Register		
0038H	Inbound Control and Status Register		
003CH	Outbound Control and Status Register		
0040H	Inbound Queue Port	2 Queue Ports	
0044H	Outbound Queue Port		
0048H	MSI Inbound Message Register		
004CH	reserved		
0050H	Intel Xscale® processor Local Memory	1004 Index Registers	
0FFCH			
1000H			MSI-X Table
	Reserved		8 Entries
17FCH	MSI-X PBA		1 Register
1800H			
1FFCH	Reserved		

Note: The MU always claim the entire 8 KBytes of the internal bus address space relative to the MU Base Address Registers. The 8 KBytes include the MU Registers, Index Registers, and MSI-X Data Structures. See more detailed descriptions under Section 311, "MU Base Address Register - MUBAR" on page 81.

B6211-01

B6211-01



Figure 42. Internal Bus Memory Map

Offsets are relative to the PMMRBAR

Offset			
4000H	Reserved	4060H	Inbound Free Head Pointer Register
4004H	Reserved	4064H	Inbound Free Tail Pointer Register
4008H	Reserved	4068H	Inbound Post Head pointer Register
400CH	Reserved	406CH	Inbound Post Tail Pointer Register
4010H	Inbound Message Register 0	4070H	Outbound Free Head Pointer Register
4014H	Inbound Message Register 1	4074H	Outbound Free Tail Pointer Register
4018H	Outbound Message Register 0	4078H	Outbound Post Head pointer Register
401CH	Outbound Message Register 1	407CH	Outbound Post Tail Pointer Register
4020H	Inbound Doorbell Register	4080H	Index Address Register
4024H	Inbound Interrupt Status Register	4084H	MU Base Address Register
4028H	Inbound Interrupt Mask Register	4088H	MU Upper Base Address Register
402CH	Outbound Doorbell Register	408CH through 4FFCH	Reserved
4030H	Outbound Interrupt Status Register	50X0 ^a H	MU MSI-X Table Message Address
4034H	Outbound Interrupt Mask Register	50X4 ^a H	MU MSI-X Table Message Upper Address
4038H	Inbound Control and Status Register	50X8 ^a H	MU MSI-X Table Message Data
403CH	Outbound Control and Status Register	50XC ^a H	MU MSI-X Table Vector Control
4040H	Reserved	5080H through 57FCH	Reserved
4044H	Reserved	5800H	MU MSI-X PBA
4048H	MSI Inbound Message Register	5804H through 5FFCH	Reserved
404CH	Reserved		
4050H	MU Configuration Register		
4054H	Queue Base Address Register		
4058H	Reserved		
405CH	Reserved		

^a X is equal to 0H, 1H, 2H, 3H, 4H, 5H, 6H, or 7H

B6212-01

Note: See [Table 264, “Message Unit Registers”](#) for more details.

[Table 258](#) provides a summary of the four messaging mechanisms used in the Messaging Unit.

Table 258. MU Summary

Mechanism	Quantity	Assert PCI Interrupt Signals?	Generate 81341 and 81342 Interrupt?
Message Registers	2 Inbound 2 Outbound	Optional	Optional
Doorbell Registers	1 Inbound 1 Outbound	Optional	Optional
Circular Queues	4 Circular Queues	Under certain conditions	Under certain conditions
Index Registers	1004 32-bit Memory Locations	No	Optional



4.2.1 Transaction Ordering

From a PCI standpoint, the Messaging Unit is a piece of the ATU and therefore must maintain ordering requirements against ATU transactions. Transaction ordering is achieved for the Index Registers, the Doorbell Register, and the Message Registers since these transactions are routed through the standard set of ATU read/write queues.

The Circular Queues (Inbound/Outbound Queue Port) are separate queue structures and therefore require ordering. The Inbound Post Queue (contains PCI writes) must be ordered against the inbound write queue of the ATU to allow the data that is represented by the Inbound Post interrupt to be written to local memory before the interrupt is delivered. See [Table 259](#) for a summary of Messaging Unit transaction ordering.

Table 259. Circular Queue Ordering Requirements

Messaging Unit Feature		Transaction Ordering Mechanism
Message Registers Doorbell Registers Index Registers		Through ATU Queues
		Through ATU Queues
		Through ATU Queues
Circular Queues	Inbound Post	Ordered Against ATU Inbound Write Queue (PMW Can Not Pass Another PMW)
	Inbound Free	No Specific Hardware Ordering
	Outbound Post	No Specific Hardware Ordering
	Outbound Free	No Specific Hardware Ordering



4.3 Message Registers

Messages can be sent and received by the 81341 and 81342 through the use of the Message Registers. When written, the message registers may cause an interrupt to be generated to either the Intel XScale® processor or the host processor. Inbound messages are sent by the host processor and received by the 81341 and 81342. Outbound messages are sent by the 81341 and 81342 and received by the host processor.

The interrupt status for outbound messages is recorded in the Outbound Interrupt Status Register. Interrupt status for inbound messages is recorded in the Inbound Interrupt Status Register.

4.3.1 Outbound Messages

When an outbound message register is written by the Intel XScale® processor, an interrupt may be generated on the **P_INTA#** interrupt pins or a message signaled interrupt is generated when MSI is enabled.

The PCI interrupt is recorded in the Outbound Interrupt Status Register. The interrupt causes the Outbound Message Interrupt bit to be set in the Outbound Interrupt Status Register. This is a Read/Clear bit that is set by the MU hardware and cleared by software.

The interrupt is cleared when an external PCI agent writes a value of 1 to the Outbound Message Interrupt bit in the Outbound Interrupt Status Register to clear the bit.

The interrupt may be masked by the mask bits in the Outbound Interrupt Mask Register.

4.3.2 Inbound Messages

When an inbound message register is written by an external PCI agent, an interrupt may be generated to the Intel XScale® processor. The interrupt may be masked by the mask bits in the Inbound Interrupt Mask Register.

The Intel XScale® processor interrupt is recorded in the Inbound Interrupt Status Register. The interrupt causes the Inbound Message Interrupt bit to be set in the Inbound Interrupt Status Register. This is a Read/Clear bit that is set by the MU hardware and cleared by software.

The interrupt is cleared when the Intel XScale® processor writes a value of 1 to the Inbound Message Interrupt bit in the Inbound Interrupt Status Register.



4.4 Doorbell Registers

There are two Doorbell Registers: the Inbound Doorbell Register and the Outbound Doorbell Register. The Inbound Doorbell Register allows external PCI agents to generate interrupts to the Intel XScale® processor. The Outbound Doorbell Register allows the Intel XScale® processor to generate a PCI interrupt. Both Doorbell Registers may generate interrupts whenever a bit in the register is set.

4.4.1 Outbound Doorbells

When the Outbound Doorbell Register is written by the Intel XScale® processor, an interrupt may be generated on the **P_INTA#** interrupt pin or a message signaled interrupt is generated when MSI is enabled. An interrupt is generated when any of the bits in the doorbell register is written to a value of 1. Writing a value of 0 to any bit does not change the value of that bit and does not cause an interrupt to be generated. Once a bit is set in the Outbound Doorbell Register, it cannot be cleared by the Intel XScale® processor.

The interrupt is recorded in the Outbound Interrupt Status Register.

The interrupt may be masked by the mask bits in the Outbound Interrupt Mask Register. When the mask bit is set for a particular bit, no interrupt is generated for that bit. The Outbound Interrupt Mask Register affects only the generation of the interrupt and not the values written to the Outbound Doorbell Register.

The interrupt is cleared when an external PCI agent writes a value of 1 to the bits in the Outbound Doorbell Register that are set. Writing a value of 0 to any bit does not change the value of that bit and does not clear the interrupt.

In summary, the Intel XScale® processor generates an interrupt and external PCI agents clear the interrupt by setting bits in the Outbound Doorbell Register.

4.4.2 Inbound Doorbells

When the Inbound Doorbell Register is written by an external PCI agent, an interrupt may be generated to the Intel XScale® processor. An interrupt is generated when any of the bits in the doorbell register is written to a value of 1. Writing a value of 0 to any bit does not change the value of that bit and does not cause an interrupt to be generated. Once a bit is set in the Inbound Doorbell Register, it cannot be cleared by any external PCI agent. The interrupt is recorded in the Inbound Interrupt Status Register.

The interrupt may be masked by the Inbound Doorbell Interrupt mask bit in the Inbound Interrupt Mask Register. When the mask bit is set for a particular bit, no interrupt is generated for that bit. The Inbound Interrupt Mask Register affects only the generation of the normal messaging unit interrupt and not the values written to the Inbound Doorbell Register. One bit in the Inbound Doorbell Register is reserved for an Error Doorbell interrupt.

The interrupt is cleared when the Intel XScale® processor writes a value of 1 to the bits in the Inbound Doorbell Register that are set. Writing a value of 0 to any bit does not change the value of that bit and does not clear the interrupt.



4.5 Circular Queues

The MU implements four circular queues. There are 2 inbound queues and 2 outbound queues. In this case, inbound and outbound refer to the direction of the flow of posted messages.

Inbound messages are either:

- *posted* messages by other processors for the Intel XScale® processor to process or
- *free* (or empty) messages that can be reused by other processors.

Outbound messages are either:

- *posted* messages by the Intel XScale® processor for other processors to process or
- *free* (or empty) messages that can be reused by the Intel XScale® processor.

Therefore, free inbound messages flow away from the 81341 and 81342 and free outbound messages flow toward the 81341 and 81342.

The four Circular Queues are used to pass messages in the following manner. The two inbound queues are used to handle inbound messages and the two outbound queues are used to handle outbound messages. One of the inbound queues is designated the Free queue and it contains inbound free messages. The other inbound queue is designated the Post queue and it contains inbound posted messages. Similarly, one of the outbound queues is designated the Free queue and the other outbound queue is designated the Post queue. [Table 260](#) contains a summary of the queues.

Table 260. Circular Queue Summary

Queue Name	Purpose	Action on PCI Interface
Inbound Post Queue	Queue for inbound messages from other processors waiting to be processed by the 81341 and 81342	Written
Inbound Free Queue	Queue for empty inbound messages from the 81341 and 81342 available for use by other processors	Read
Outbound Post Queue	Queue for outbound messages from the 81341 and 81342 that are being posted to the other processors	Read
Outbound Free Queue	Queue for empty outbound messages from other processors available for use by the 81341 and 81342	Written

The two outbound queues allow the Intel XScale® processor to post outbound messages in one queue and to receive free messages returning from the host processor. The Intel XScale® processor posts outbound messages, the host processor receives the posted message and when it is finished with the message, places it back on the outbound free queue for reuse by the Intel XScale® processor.

The two inbound queues allow the host processor to post inbound messages for the 81341 and 81342 in one queue and to receive free messages returning from the 81341 and 81342. The host processor posts inbound messages, the Intel XScale® processor receives the posted message and when it is finished with the message, places it back on the inbound free queue for reuse by the host processor.

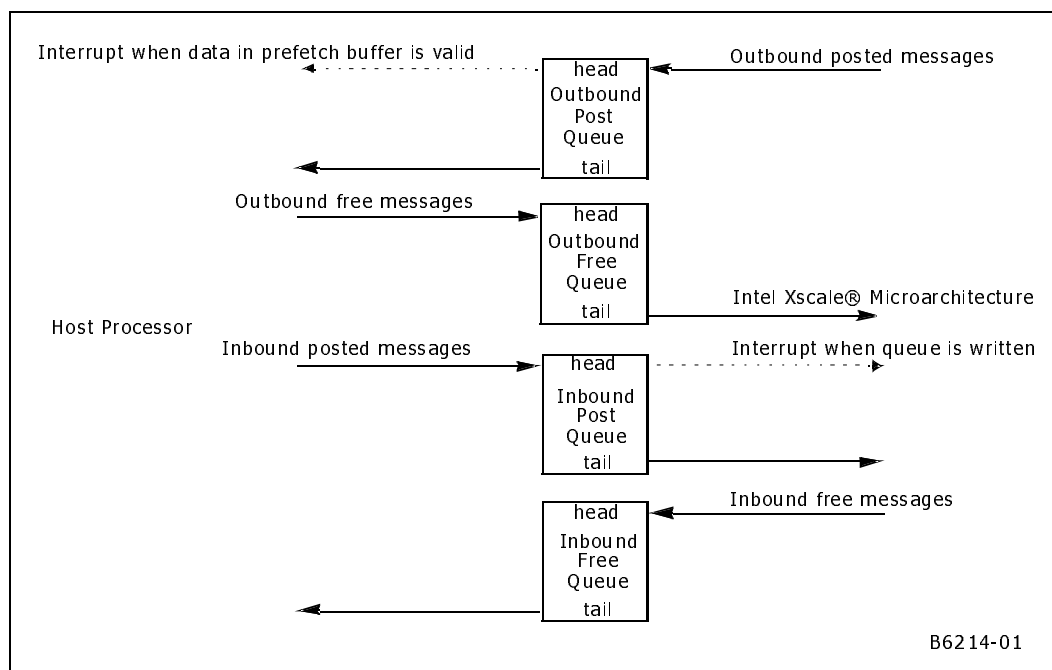
Warning: External PCI write transactions to the Inbound Post Queue or the Outbound Free Queue with NULL byte enables (all byte enables deasserted) causes the Head Pointer to increment.



Figure 43 provides an overview of the Circular Queue operation.

The circular queues are accessed by external PCI agents through two port locations in the PCI address space: Inbound Queue Port and Outbound Queue Port. The Inbound Queue Port is used by external PCI agents to read the Inbound Free Queue and write the Inbound Post Queue. The Outbound Queue Port is used by external PCI agents to read the Outbound Post Queue and write the Outbound Free Queue.

Figure 43. Overview of Circular Queue Operation



The data storage for the circular queues must be provided by the 81341 and 81342 local memory. The base address of the circular queues is contained in the Queue Base Address Register (Section 4.9.13, “Queue Base Address Register - QBAR” on page 438). Each entry in the queue is a 32-bit data value. Each read from or write to the queue may access only one queue entry. Multi-word accesses to the circular queues are not allowed.

Each circular queue has a head pointer and a tail pointer. The pointers are offsets from the Queue Base Address. Writes to a queue occur at the head of the queue and reads occur from the tail. The head and tail pointers are incremented by either the Intel XScale® processor or the Messaging Unit hardware. Which unit maintains the pointer is determined by the writer of the queue. More details about the pointers are given in the queue descriptions below. The pointers are incremented after the queue access. Both pointers wrap around to the first address of the circular queue when they reach the circular queue size.

The Messaging Unit generates an interrupt to the Intel XScale® processor or generate a PCI interrupt under certain conditions. In general, when a Post queue is written, an interrupt is generated to notify the receiver that a message was posted.

The size of each circular queue can range from 4K entries (16 Kbytes) to 64K entries (256 Kbytes). All four queues must be the same size and may be contiguous. Therefore, the total amount of local memory needed by the circular queues ranges from 64 Kbytes to 1 Mbytes. The Queue size is determined by the Queue Size field in the MU Configuration Register.



There is one base address for all four queues. It is stored in the Queue Base Address Register (QBAR). The starting addresses of each queue is based on the Queue Base Address and the Queue Size field. Table 261 shows an example of how the circular queues should be set up based on the *Intelligent I/O (I₂O) Architecture Specification*. Other ordering of the circular queues is possible.

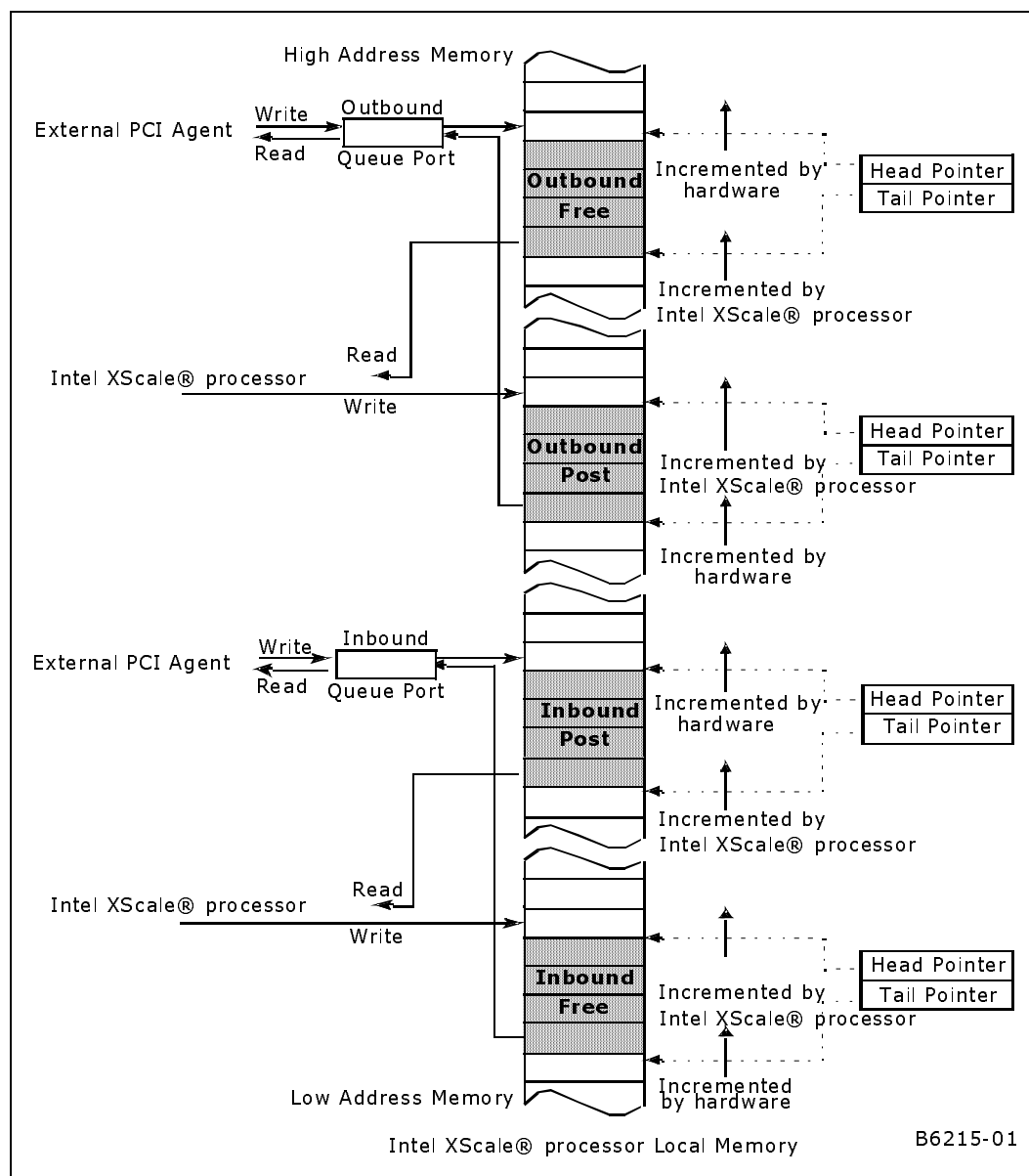
Table 261. Queue Starting Addresses

Queue	Starting Address
Inbound Free Queue	QBAR
Inbound Post Queue	QBAR + Queue Size
Outbound Post Queue	QBAR + 2 * Queue Size
Outbound Free Queue	QBAR + 3 * Queue Size



Figure 44 provides a more detailed diagram of the usage of the Circular Queues.

Figure 44. Circular Queue Operation



4.5.1 Inbound Free Queue

The Inbound Free Queue holds free inbound messages placed there by the Intel XScale® processor for other processors to use. This queue is read from the queue tail by external host I/O interface agents on the 81341 and 81342's internal bus through the Address Translation Unit (See [Chapter 2.0, "Address Translation Unit \(PCI-X\)"](#) or [Chapter 3.0, "Address Translation Unit \(PCI Express\)"](#) for more details on inbound ATU addressing and the ATU). It is written to the queue head by the Intel XScale® processor. The tail pointer is maintained by the MU hardware. The head pointer is maintained by the Intel XScale® processor.

For a host I/O interface read transaction that accesses the Inbound Queue Port, the MU attempts to read the data at the local memory address in the Inbound Free Tail Pointer. When the queue is not empty (head and tail pointers are not equal) or full (head and tail pointers are equal but the head pointer was last written by software¹⁰), the data is returned. When the queue is empty (head and tail pointers are equal and the tail pointer was last updated by hardware¹¹), the value of -1 (FFFF.FFFFH) is returned. When the queue was not empty and the MU succeeded in returning the data at the tail, the MU hardware must increment the value in the Inbound Free Tail Pointer Register.

To reduce latency for the host I/O interface read access, the MU implements a prefetch mechanism to anticipate accesses to the Inbound Free Queue. The MU hardware prefetches the data at the tail of the Inbound Free Queue and load it into an internal prefetch register. When the internal bus read access occurs, the data is read directly from the prefetch register.

The prefetch mechanism loads a value of -1 (FFFF.FFFFH) into the prefetch register when the head and tail pointers are equal and the queue is empty. In order to update the prefetch register when messages are added to the queue and it becomes non-empty, the prefetch mechanism automatically starts a prefetch when the prefetch register contains FFFF.FFFFH and the Inbound Free Head Pointer Register is written. The Intel XScale® processor needs to update the Inbound Free Head Pointer Register when it adds messages to the queue.

A prefetch must appear atomic from the perspective of the ATU. When a prefetch is started, any internal bus transaction that attempts to access the Inbound Free Queue is signalled a Retry until the prefetch is completed.

The Intel XScale® processor may place messages in the Inbound Free Queue by writing the data to the local memory location pointed to by the Inbound Free Head Pointer Register. The processor must then increment the Inbound Free Head Pointer Register.

10. The Inbound Free Queue Head Pointer is only managed by **software** using the "Inbound Free Head Pointer Register - IFHPR" during normal operation and initialization.

11. During normal operation, the Inbound Free Queue Tail Pointer is only managed by **hardware**. **Software** can also update the Tail Pointer using the "Inbound Free Tail Pointer Register - IFTPR" during initialization. However, the Inbound Free Queue logic does not make a distinction on whether the Tail Pointer is updated using hardware or software.



4.5.2 Inbound Post Queue

The Inbound Post Queue holds posted messages placed there by other processors for the Intel XScale® processor to process. This queue is read from the queue tail by the Intel XScale® processor. It is written to the queue head by external host I/O interface agents on the 81341 and 81342's internal bus through the Address Translation Unit (See [Chapter 2.0, "Address Translation Unit \(PCI-X\)"](#) or [Chapter 3.0, "Address Translation Unit \(PCI Express\)"](#) for more details on inbound ATU addressing and the ATU). The tail pointer is maintained by the Intel XScale® processor. The head pointer is maintained by the MU hardware.

For a host I/O interface write transaction that accesses the Inbound Queue Port, the MU writes the data to the local memory location address in the Inbound Post Head Pointer Register.

When the data written to the Inbound Queue Port is written to local memory, the MU hardware increments the Inbound Post Head Pointer Register.

An Intel XScale® processor interrupt may be generated when the Inbound Post Queue is written. The Inbound Post Queue Interrupt bit in the Inbound Interrupt Status Register indicates the interrupt status. The interrupt is cleared when the Inbound Post Queue Interrupt bit is cleared. The interrupt can be masked by the Inbound Interrupt Mask Register. Software must be aware of the state of the Inbound Post Queue Interrupt Mask bit to insure that the full condition is recognized by the core processor. In addition, to insure that the queue does not get overwritten, software must process messages from the tail of the queue before incrementing the tail pointer and clearing this interrupt. Once cleared, an interrupt is NOT generated when the head and tail pointers remain unequal (i.e. queue status is Not Empty). Only a new message posting in the inbound queue generates a new interrupt. Therefore, when software leaves any unprocessed messages in the post queue when the interrupt is cleared, software must retain the information that the Inbound Post queue status.

From the time that the host I/O interface write transaction is received on the internal bus by the MU until the data is written in local memory and the Inbound Post Head Pointer Register is incremented, any internal bus transaction that attempts to access the Inbound Post Queue Port is signalled a Retry.

The Intel XScale® processor may read messages from the Inbound Post Queue by reading the data from the local memory location pointed to by the Inbound Post Tail Pointer Register. The Intel XScale® processor must then increment the Inbound Post Tail Pointer Register. When the Inbound Post Queue is full (head¹² and tail¹³ pointers are equal and the head pointer was last updated by hardware), the hardware retries any write until a slot in the queue becomes available. A slot in the post queue becomes available by the Intel XScale® processor incrementing the tail pointer.

-
12. During normal operation, the Inbound Post Queue Head Pointer is only managed by **hardware**. **Software** can also update the Head Pointer using the "Inbound Post Head Pointer Register - IPHPR" during initialization. However, the Inbound Post Queue logic does not make a distinction on whether the Head Pointer is updated using hardware or software.
 13. The Inbound Post Queue Tail Pointer is only managed by **software** using the "Inbound Post Tail Pointer Register - IPTPR" during normal operation and initialization.

4.5.3 Outbound Post Queue

The Outbound Post Queue holds outbound posted messages placed there by the Intel XScale® processor for other processors to process. This queue is read from the queue tail by external host I/O interface agents on the 81341 and 81342's internal bus through the Address Translation Unit (See [Chapter 2.0, "Address Translation Unit \(PCI-X\)"](#) or [Chapter 3.0, "Address Translation Unit \(PCI Express\)"](#) for more details on inbound ATU addressing and the ATU). It is written to the queue head by the Intel XScale® processor. The tail pointer is maintained by the MU hardware. The head pointer is maintained by the Intel XScale® processor.

For a host I/O interface read transaction that accesses the Outbound Queue Port, the MU attempts to read the data at the local memory address in the Outbound Post Tail Pointer Register. When the queue is not empty (head and tail pointers are not equal) or full (head and tail pointers are equal but the head pointer was last written by software¹⁴), the data is returned. When the queue is empty (head and tail pointers are equal and the tail pointer was last updated by hardware¹⁵), the value of -1 (FFFF.FFFFH) is returned. When the queue was not empty and the MU succeeded in returning the data at the tail, the MU hardware must increment the value in the Outbound Post Tail Pointer Register.

To reduce latency for the host I/O interface read access, the MU implements a prefetch mechanism to anticipate accesses to the Outbound Post Queue. The MU hardware prefetches the data at the tail of the Outbound Post Queue and load it into an internal prefetch register. When the internal bus read access occurs, the data is read directly from the prefetch register.

The prefetch mechanism loads a value of -1 (FFFF.FFFFH) into the prefetch register when the head and tail pointers are equal and the queue is empty. In order to update the prefetch register when messages are added to the queue and it becomes non-empty, the prefetch mechanism automatically starts a prefetch when the prefetch register contains FFFF.FFFFH and the Outbound Post Head Pointer Register is written. The Intel XScale® processor needs to update the Outbound Post Head Pointer Register when it adds messages to the queue.

A prefetch must appear atomic from the perspective of the ATU. When a prefetch is started, any internal bus transaction that attempts to access the Outbound Post Queue is signalled a Retry until the prefetch is completed.

A host I/O interface interrupt may be generated when data in the prefetch buffer is valid. When the prefetch queue is clear, no interrupt is generated. The Outbound Post Queue Interrupt bit in the Outbound Interrupt Status Register shall indicate the status of the prefetch buffer data and therefore the interrupt status. The interrupt is cleared when any prefetched data has been read from the Outbound Queue Port. The interrupt can be masked by the Outbound Interrupt Mask Register.

The Intel XScale® processor may place messages in the Outbound Post Queue by writing the data to the local memory address in the Outbound Post Head Pointer Register. The processor must then increment the Outbound Post Head Pointer Register.

14. The Outbound Post Queue Head Pointer is only managed by **software** using the "Outbound Post Head Pointer Register - OPHPR" during normal operation and initialization.

15. During normal operation, the Outbound Post Queue Tail Pointer is only managed by **hardware**. **Software** can also update the Tail Pointer using the "Outbound Post Tail Pointer Register - OPTPR" during initialization. However, the Outbound Post Queue logic does not make a distinction on whether the Tail Pointer is updated using hardware or software.



4.5.4 Outbound Free Queue

The Outbound Free Queue holds free messages placed there by other processors for the Intel XScale® processor to use. This queue is read from the queue tail by the Intel XScale® processor. It is written to the queue head by external host I/O interface agents on the 81341 and 81342's internal bus through the Address Translation Unit (See [Chapter 2.0, "Address Translation Unit \(PCI-X\)"](#) or [Chapter 3.0, "Address Translation Unit \(PCI Express\)"](#) for more details on inbound ATU addressing and the ATU). The tail pointer is maintained by the Intel XScale® processor. The head pointer is maintained by the MU hardware.

For a host I/O interface write transaction that accesses the Outbound Queue Port, the MU writes the data to the local memory address in the Outbound Free Head Pointer Register. When the data written to the Outbound Queue Port is written to local memory, the MU hardware increments the Outbound Free Head Pointer Register.

When the head¹⁶ pointer and the tail¹⁷ pointer become equal and the queue is full, the MU may signal an interrupt to the Intel XScale® processor to register the queue full condition. This interrupt is recorded in the Inbound Interrupt Status Register. The interrupt is cleared when the Outbound Free Queue Full Interrupt bit is cleared and not by writing to the head or tail pointers. The interrupt can be masked by the Inbound Interrupt Mask Register. Software must be aware of the state of the Outbound Free Queue Interrupt Mask bit to insure that the full condition is recognized by the core processor.

From the time that a host I/O interface write transaction is received on the internal bus by the MU until the data is written in local memory and the Outbound Free Head Pointer Register is incremented, any internal bus transaction that attempts to access the Outbound Free Queue Port is signalled a retry.

The Intel XScale® processor may read messages from the Outbound Free Queue by reading the data from the local memory address in the Outbound Free Tail Pointer Register. The processor must then increment the Outbound Free Tail Pointer Register. When the Outbound Free Queue is full, the hardware must retry any write until a slot in the queue becomes available.

-
16. During normal operation, the Outbound Free Queue Head Pointer is only managed by **hardware**. **Software** can also update the Head Pointer using the "Outbound Free Head Pointer Register - OFHPR" during initialization. However, the Outbound Free Queue logic does not make a distinction on whether the Head Pointer is updated using hardware or software.
17. The Outbound Free Queue Tail Pointer is only managed by **software** using the "Outbound Free Tail Pointer Register - OFTPR" during normal operation and initialization.



Table 262. Circular Queue Summary

Queue Name	Host I/O Interface Port	Generate Host I/O Interface Interrupt	Generate Intel XScale® processor Interrupt	Head Pointer maintained by	Tail Pointer maintained by
Inbound Post Queue	Inbound Queue Port	No	Yes, when queue is written	MU hardware	Intel XScale® microarchitecture
Inbound Free Queue		No	No	Intel XScale® microarchitecture	MU hardware
Outbound Post Queue	Outbound Queue Port	Yes, when data in prefetch buffer is valid	No	Intel XScale® microarchitecture	MU hardware
Outbound Free Queue		No	Yes, when the queue is full	MU hardware	Intel XScale® microarchitecture

Table 263. Circular Queue Status Summary

Queue Name	Queue Status	Head & Tail Pointer	Last Pointer Update
Inbound Post Queue & Outbound Free Queue	Empty	Equal	Tail pointer last updated by software ^a
	Not Empty	Not Equal	N/A
	Full	Equal	Head pointer last updated by hardware ^b
Inbound Free Queue & Outbound Post Queue	Empty	Equal	Tail pointer last updated by hardware ^c
	Not Empty	Not Equal	
	Full	Equal	Head pointer last updated by software ^d

- The Inbound Post Queue and Outbound Free Queue Tail Pointers are only managed by **software** respectively using the “Inbound Post Tail Pointer Register - IPTPR” and “Outbound Free Tail Pointer Register - OFTPR” during normal operation and initialization.
- During normal operation, the Inbound Post Queue and Outbound Free Queue Head Pointers are only managed by **hardware**. **Software** can also update the Head Pointers respectively using the “Inbound Post Head Pointer Register - IPHPR” and “Outbound Free Head Pointer Register - OFHPR” during initialization. However, the Inbound Post Queue and Outbound Free Queue logic does not make a distinction on whether the Head Pointers are updated using hardware or software.
- During normal operation, the Inbound Free Queue and the Outbound Post Queue Tail Pointers are only managed by **hardware**. **Software** can also update the Tail Pointers respectively using the “Inbound Free Tail Pointer Register - IFTPR” and “Outbound Post Tail Pointer Register - OPTPR” during initialization. However, the Inbound Free Queue and Outbound Post Queue logic does not make a distinction on whether the Tail Pointers are updated using hardware or software.
- The Inbound Free Queue and the Outbound Post Queue Head Pointers are only managed by **software** respectively using the “Inbound Free Head Pointer Register - IFHPR” and “Outbound Post Head Pointer Register - OPHPR” during normal operation and initialization.



4.6 Index Registers

The Index Registers are a set of 1004 registers that when written by an external host I/O interface agent can generate an interrupt to the Intel XScale® processor. These registers are for inbound messages only. The interrupt is recorded in the Inbound Interrupt Status Register.

The storage for the Index Registers is allocated from the 81341 and 81342 local DDR memory. Write accesses to the Index Registers write the data to local DDR memory. Read accesses to the Index Registers read the data from local DDR memory. The local DDR memory used for the Index Registers ranges from MU Base Address Registers +50H to the MU Base Address Registers + FFFH. The MU Base Address registers (MUBAR and MUUBAR) must be programmed properly in order to map the MU space within DDR memory space, and the DDR Memory space must be within the ATU translation window. The MU Base Address Registers allow the index registers to be placed in any 8-KByte space of the Host I/O Interface Address Translation window.

The address of the first write access is stored in the Index Address Register. This register is written during the earliest write access and provides a means to determine which Index Register was written. Once updated by the MU, the Index Address Register is not updated until the Index Register Interrupt bit in the Inbound Interrupt Status Register is cleared. When the interrupt is cleared, the Index Address Register is re-enabled and stores the address of the next Index Register write access.

Writes by the Intel XScale® processor to the local memory used by the Index Registers does not cause an interrupt and does not update the Index Address Register.

The index registers can be accessed with multi-word transactions.

Warning: The MU Base Address Register (MUBAR and MUUBAR) must be programmed by software for proper operation of the MU. Using the MUBAR and MUUBAR, the MU can be relocated to any address space on the internal bus. However, the MU must be mapped in the ATU translated address space. In addition, the MU must overlap the DDR memory address space in order for the Index Registers to be functional.



4.7 Messaging Unit Error Conditions

The Messaging Unit, like the ATU, encounters error conditions on the host I/O interface as well as the internal bus interface. As a host I/O interface target, all host I/O interface errors are captured and recorded in the ATU Status Register and can be masked using the ATU mechanisms. Refer to [Chapter 2.0, "Address Translation Unit \(PCI-X\)"](#) or [Chapter 3.0, "Address Translation Unit \(PCI Express\)"](#) for further details.



4.8 Message-Signaled Interrupts

4.8.1 MSI Capability Structure

When a host processor enables Message-Signaled Interrupts (MSI) on the 81341 and 81342 ATU function, the ATU function (MU) is responsible to signal interrupt to the host via a host I/O interface write instead of the assertion of the **P_INTA#** output pin.

The *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a states that “PCI-X devices that generate interrupts are required to support message-signaled interrupts, as defined by the *PCI Local Bus Specification*, Revision 2.2 and must support a 64-bit message address.” “Devices that require interrupts in systems that do not support message-signaled interrupts, must implement interrupt pins.” Thus, the 81341 and 81342 needs to implement both wired and message-signaled interrupt delivery mechanisms.

In support of MSI, the 81341 and 81342 implements the MSI capability structure. The capability structure includes the “[Message Control Register - Message_Control](#)” on page 452, the “[Message Address Register - Message_Address](#)” on page 453, the “[Message Upper Address Register - Message_Upper_Address](#)” on page 454 and the “[Message Data Register- Message_Data](#)” on page 455.

During system initialization, the configuration software for an MSI system reads the Message Control Register to determine that the 81341 and 81342 supports a 64-bit Message Address, and that it is capable of generating two unique interrupt messages.

After gathering this data from all of the MSI capable devices in the system, the configuration software decides where to initialize the Message Address and how many unique messages each MSI capable device is allowed. Then, software writes the Message Address Registers (and the Message Upper Address Registers when Message Address is above the 4G address boundary¹⁸), and the Message Data Register. This system specified data is used to route the interrupt request message to the appropriate entry in a host processor Local APIC table.

Configuration of MSI completes with a write to the Message Control Register which includes an update to the Multiple Message Enable field and the MSI enable bit of each device. This informs the device how many unique messages (Local APIC table entries) have been allocated for exclusive use by that device and enable that device for MSI. Device hardware is required to handle allocation of fewer unique interrupt messages than requested by the Multiple Message Capable field.

The 81341 and 81342 is able to handle generating only one message, even though the device is capable of generating two unique messages. When two unique messages are enabled, one message is reserved for Outbound Post Queue Interrupt, the other message represents all of Outbound Doorbell and Outbound Message Interrupts. When only one message is enabled, all interrupts are represented by a single message. Interrupt handler software needs to read the 81341 and 81342 Outbound Interrupt Status Register to determine the cause of the interrupt when more than one source is represented by a single message.

To signal an Outbound Interrupt with MSI enabled, the 81341 and 81342 creates an outbound write transaction using the Message Address and the Message Data. When two unique messages are enabled, the lowest order bit of the Message Data is modified by hardware so that the host processor can distinguish between them.

18. When host software writes the Message Upper address register to a non-zero value, device hardware uses a write transaction with a Dual Address Cycle (DAC) to present the full 64-bit address to the bus.

4.8.2 MSI-X Capability and Table Structures

Similar to MSI Capability, when a host processor enables Message-Signaled Interrupts (MSI-X) on the 81341 and 81342 ATU function, the ATU function (MU) is responsible to signal interrupt to the host via a PCI write instead of the assertion of the **P_INTA#** output pin.

The *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a states that “PCI-X devices that generate interrupts are required to support message-signaled interrupts, as defined by the *PCI Local Bus Specification*, Revision 2.2 and must support a 64-bit message address.” “Devices that require interrupts in systems that do not support message-signaled interrupts, must implement interrupt pins.” Thus, the 81341 and 81342 needs to implement both wired and message-signaled interrupt delivery mechanisms.

In support of MSI-X, the 81341 and 81342 implements the MSI-X capability structure. The capability structure includes the “[MSI-X Capability Identifier Register - MSI-X_Cap_ID](#)” on page 456, the “[MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr](#)” on page 457, the “[MSI-X Message Control Register - MSI-X_MCR](#)” on page 458, the “[MSI-X Table Offset Register — MSI-X_Table_Offset](#)” on page 459 and the “[MSI-X Pending Bit Array Offset Register - MSI-X_PBA_Offset](#)” on page 460.

During system initialization, the configuration software for an MSI-X system reads the Message Control Register to determine that the 81341 and 81342 is capable of generating eight unique interrupt messages by reading the MSI-X table size. The configuration software also reads the MSI-X Table offset Register and MSI-X Pending Bits Array Register to determine the locations these structures.

After gathering this data from all of the MSI-X capable devices in the system, the configuration software decides how to initialize the MSI-X Table by writing the Message Address Registers (and the Message Upper Address Registers when Message Address is above the 4G address boundary¹⁹), the Message Data Registers, and the Vector Control Registers in order to unmask a Table entry. This system specified data is used to route the interrupt request message to the appropriate entry in a host processor’s Local APIC table.

Configuration of MSI-X completes with a write to the MSI-X Message Control Register which includes an update to the Multiple Message Enable field.

The MU on 81341 and 81342 is able to handle generating up to eight unique messages, representing all of Outbound Doorbell Interrupts. Each outbound interrupt on the MU has a unique MSI-X message associated to it. The MU can also be setup to generate only a single MSI-X message. A single MSI-X message is generated when the MU MSI-X Single Message Vector bit is set in the “[MU MSI-X Control Register X — MMCRx](#)” on page 461.

Note:

The MU can signal an interrupt (MU MSI-X Table Write Interrupt) to the 81341 and 81342 interrupt controller when the Host processor writes any entry in the MU MSI-X Table. The MU MSI-X Table includes the following fields: the Message Address Registers, the Message Upper Address Registers, the Message Data Registers, and the Vector Control Registers. The interrupt can be enabled or disabled using the “[Inbound Interrupt Mask Register - IIMR](#)”, and interrupt is posted in the “[Inbound Interrupt Status Register - IISR](#)”.

19. When host software writes the Message Upper address register to a non-zero value, device hardware uses a write transaction with a Dual Address Cycle (DAC) to present the full 64-bit address to the bus.



To signal an Outbound Interrupt with MSI-X enabled, the 81341 and 81342 creates an outbound write transaction using the Message Address and the Message Data of the associated entry. On 81341 and 81342, entry 0 of the MSI-X Table is assigned to bit OISR[0], entry 1 is assigned to bit OISR[1] and so on. For example, entry 7 of the MSI-X Table is assigned to bit OISR[7].

Note: Entry 2 is assigned to bits OSIR[2] and OISR[31]. For example, the Outbound Doorbell Register and the Firmware Interrupt bit in the Outbound Control and Status Register.

Note: When host software enables MSI, a Messaging Unit Interrupt does not result in the assertion of the **P_INTx#** output pin. However, all the **P_INT[A:D]#** pins are functional for steering of interrupts from other PCI devices that may not be MSI capable.

MSI-X Table and Pending Bits Array are mapped relative to the PCI/Host interface (Figure 45). The MU registers are located in the first 4-KByte of the 8-KByte address space claimed by the MU, whereas the MSI-X Table is located at a 4-KByte offset and the MSI-X PBA is located at a 6-KByte offset relative to the MU Base Address. Note that the MU Base Address register must be programmed such that it overlaps the address space defined by the ATU Translate and ATU Limit Registers. For example, the MU 8-KByte window must overlap onto the ATU Translation Window.

Figure 45. MSI-X Table and PBA Address Mapping Layout relative to the Host Interface

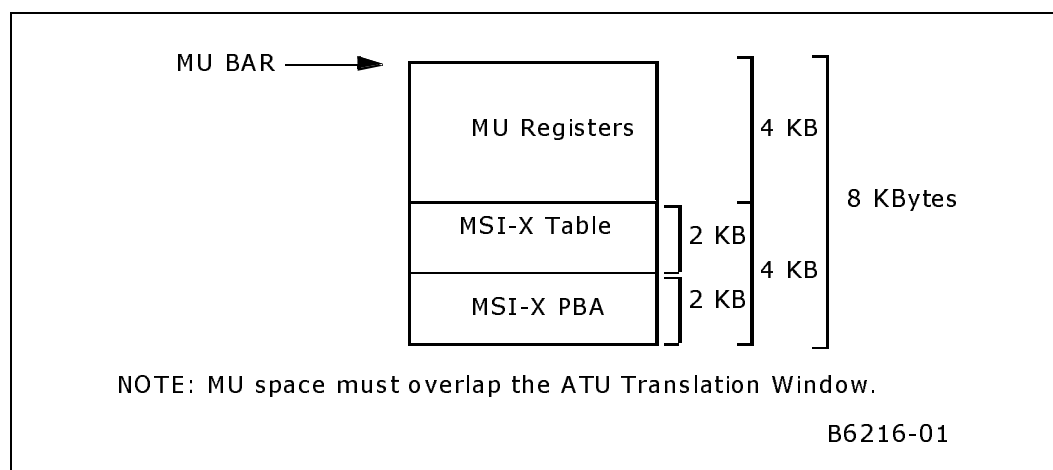
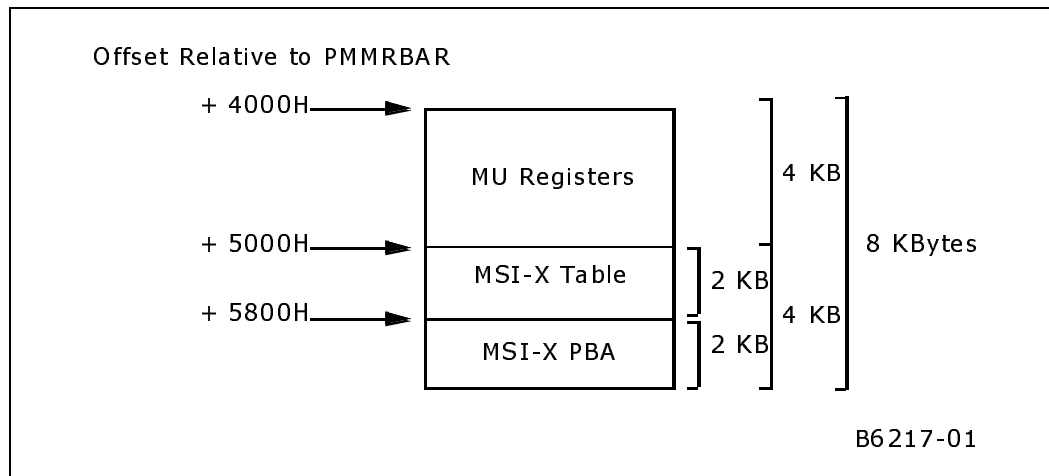


Figure 46 shows how the MSI-X Table and Pending Bits Array are mapped from the host I/O interface. The MU registers are located in the first 4-KByte of the 8-KByte address space.

Figure 46. MSI-X Table and PBA Address Mapping Layout relative to the Internal Bus



4.8.3 Level-Triggered Versus Edge-Triggered Interrupts

When MSI and MSI-X are disabled, the **P_INTA#** pin remains asserted and pended to the host when **any** of the MU interrupt sources requires service (Outbound Post Queue, Outbound Doorbell and Outbound Message). Since the PCI pin signaled interrupt is **level-triggered**, the interrupt service routine does not drop out of the service routine until the interrupt signal is deasserted. This insures that an interrupt is not missed.

MSI interrupts are inherently edge-triggered, in that an interrupt is only pended to the host as a write event when any of the MU interrupt sources requires service.

Note: Bit[10] (Interrupt Disable bit) of the ATUs ATU Command Register (ATUCMD) must be cleared for the **P_INTA#** interrupt to be generated to the Host processor. Bit[3] (Interrupt Status bit) of the ATU's ATU Status Register (ATUSR) is not affected by the state of bit[10] of the ATUCMD.



4.9 Register Definitions

The following registers are located in the Host I/O Interface address space and in the Peripheral Memory-Mapped Register (PMMR) address space. They are accessible through host I/O interface bus transactions and through Intel XScale® processor internal bus accesses. In the Host I/O Interface address space, they are mapped into the first 80 bytes of the inbound address window of the ATU.

- Inbound Message 0 Register
- Inbound Message 1 Register
- Outbound Message 0 Register
- Outbound Message 1 Register
- Inbound Doorbell Register
- Inbound Interrupt Status Register
- Inbound Interrupt Mask Register
- Outbound Doorbell Register
- Outbound Interrupt Status Register
- Outbound Interrupt Mask Register
- Inbound Reset Control and Status Register
- Outbound Reset Control and Status Register
- [MSI Inbound Message Register](#)

The following registers are located in the Peripheral Memory-Mapped Register (PMMR) address space as described in [Chapter 21.0, "Peripheral Registers"](#).

- MU Configuration Register
- Queue Base Address Register
- Inbound Free Head Pointer Register
- Inbound Free Tail Pointer Register
- Inbound Post Head Pointer Register
- Inbound Post Tail Pointer Register
- Outbound Free Head Pointer Register
- Outbound Free Tail Pointer Register
- Outbound Post Head Pointer Register
- Outbound Post Tail Pointer Register
- Index Address Register
- MU Base Address Register
- MU Upper Base Address Register

Reading or writing a register that is reserved is undefined.

**Table 264. Message Unit Registers**

Internal Bus Address Offset	Section, Register Name - Acronym (Page)
4010H	Section 4.9.1, "Inbound Message Register - IMRx" on page 427
4014H	Section 4.9.1, "Inbound Message Register - IMRx" on page 427
4018H	Section 4.9.2, "Outbound Message Register - OMRx" on page 427
401CH	Section 4.9.2, "Outbound Message Register - OMRx" on page 427
4020H	Section 4.9.3, "Inbound Doorbell Register - IDR" on page 428
4024H	Section 4.9.4, "Inbound Interrupt Status Register - IISR" on page 429
4028H	Section 4.9.5, "Inbound Interrupt Mask Register - IIMR" on page 430
402CH	Section 4.9.6, "Outbound Doorbell Register - ODR" on page 431
4030H	Section 4.9.7, "Outbound Interrupt Status Register - OISR" on page 432
4034H	Section 4.9.8, "Outbound Interrupt Mask Register - OIMR" on page 433
4038H	Section 4.9.9, "Inbound Reset Control and Status Register - IRCSR" on page 434
403CH	Section 4.9.10, "Outbound Reset Control and Status Register - ORCSR" on page 435
4048H	Section 4.9.11, "MSI Inbound Message Register — MIMR" on page 436
4050H	Section 4.9.12, "MU Configuration Register - MUCR" on page 437
4054H	Section 4.9.13, "Queue Base Address Register - QBAR" on page 438
4060H	Section 4.9.14, "Inbound Free Head Pointer Register - IFHPR" on page 439
4064H	Section 4.9.15, "Inbound Free Tail Pointer Register - IFTPR" on page 439
4068H	Section 4.9.16, "Inbound Post Head Pointer Register - IPHPR" on page 440
406CH	Section 4.9.17, "Inbound Post Tail Pointer Register - IPTPR" on page 440
4070H	Section 4.9.18, "Outbound Free Head Pointer Register - OFHPR" on page 441
4074H	Section 4.9.19, "Outbound Free Tail Pointer Register - OFTPR" on page 441
4078H	Section 4.9.20, "Outbound Post Head Pointer Register - OPHPR" on page 442
407CH	Section 4.9.21, "Outbound Post Tail Pointer Register - OPTPR" on page 442
4080H	Section 4.9.22, "Index Address Register - IAR" on page 443
4084H	Section 4.9.23, "MU Base Address Register - MUBAR" on page 444
4088H	Section 4.9.24, "MU Upper Base Address Register - MUUBAR" on page 445
408CH - 4FFCH	Reserved.
50X0H ¹	Section 4.9.25, "MU MSI-X Table Message Address Registers - M_MT_MAR[0:7]" on page 446
50X4H ²	Section 4.9.26, "MU MSI-X Table Message Upper Address Registers - M_MT_MUAR[0:7]" on page 447
50X8H ³	Section 4.9.27, "MU MSI-X Table Message Data Registers - M_MT_MDR[0:7]" on page 448
50XCH ⁴	Section 4.9.28, "MU MSI-X Table Message Vector Control Registers - M_MT_MVCR[0:7]" on page 449
5080H - 57FCH	Reserved.
5800H	Section 4.9.29, "MU MSI-X Pending Bits Array Register - M_MPBAR" on page 450
5804H - 5FFCH	Reserved.

Notes:

1. X is equal to 0H, 1H, 2H, 3H, 4H, 5H, 6H, or 7H.
2. X is equal to 0H, 1H, 2H, 3H, 4H, 5H, 6H, or 7H.
3. X is equal to 0H, 1H, 2H, 3H, 4H, 5H, 6H, or 7H.
4. X is equal to 0H, 1H, 2H, 3H, 4H, 5H, 6H, or 7H.



4.9.1 Inbound Message Register - IMRx

There are two Inbound Message Registers: IMR0 and IMR1. When the IMR register is written, an interrupt to the Intel XScale® processor may be generated. The interrupt is recorded in the Inbound Interrupt Status Register and may be masked by the Inbound Message Interrupt Mask bit in the Inbound Interrupt Mask Register.

Table 265. Inbound Message Register - IMRx

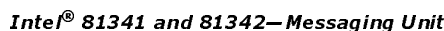
		The diagram shows a 32-bit register divided into two 16-bit sections. The top section is labeled 'IOP Attributes' and the bottom section is labeled 'PCI Attributes'. Both sections are divided into 16-bit words, each labeled 'rw' (Read/Write). Bit positions 31, 28, 24, 20, 16, 12, 8, and 4 are marked at the top of the register.																																	
MU/PCI Base Address Offset		internal bus address offset																Attribute Legend:																	
IMR0: 0010H		IMR0: 4010H																RV = Reserved																RC = Read Clear	
IMR1: 0014H		IMR1: 4014H																PR = Preserved																RO = Read Only	
																		RS = Read/Set																NA = Not Accessible	
Bit	Default	Description																																	
31:00	0000 0000H	Inbound Message - This is a 32-bit message written by an external Host I/O Interface agent. When written, an interrupt to the Intel XScale® processor may be generated.																																	

4.9.2 Outbound Message Register - OMRx

There are two Outbound Message Registers: OMR0 and OMR1. When the OMR register is written, a Host I/O Interface interrupt may be generated. The interrupt is recorded in the Outbound Interrupt Status Register and may be masked by the Outbound Message Interrupt Mask bit in the Outbound Interrupt Mask Register.

Table 266. Outbound Message Register - OMRx

IOP Attributes PCI Attributes																															
MU/PCI Base Address Offset								internal bus address offset								Attribute Legend:								RW = Read/Write							
OMR0: 0018H								OMR0: 4018H								RV = Reserved								RC = Read Clear							
OMR1: 001CH								OMR1: 401CH								PR = Preserved								RO = Read Only							
																RS = Read/Set								NA = Not Accessible							
Bit	Default	Description																													
31:00	00000000H	Outbound Message - This is 32-bit message written by the Intel XScale® processor. When written, an interrupt may be generated on the PCI Interrupt pin determined by the ATU Interrupt Pin Register.																													



Inbound Doorbell Register - IDR

The Inbound Doorbell Register (IDR) is used to generate interrupts to the Intel XScale® processor. Bit 31 is reserved for generating an Error Doorbell interrupt. When bit 31 is set, an Error interrupt may be generated to the Intel XScale® processor. All other bits, when set, cause the Normal Messaging Unit interrupt line of the Intel XScale® processor to be asserted, when the interrupt is not masked by the Inbound Doorbell Interrupt Mask bit in the Inbound Interrupt Mask Register. The bits in the IDR register can only be set by an external Host I/O Interface agent and can only be cleared by the Intel XScale® processor.

Table 267. Inbound Doorbell Register - IDR

MU/PCI Base Address Offset: 0020H
 internal bus address offset: 4020H

Attribute Legend:
 RV = Reserved
 RC = Read Clear
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



4.9.4 Inbound Interrupt Status Register - IISR

The Inbound Interrupt Status Register (IISR) contains hardware interrupt status. It records the status of Intel XScale® processor interrupts generated by the Message Registers, Doorbell Registers, and the Circular Queues. All interrupts are routed to the Normal Messaging Unit interrupt input of the Intel XScale® processor, except for the Error Doorbell Interrupt and the Outbound Free Queue Full interrupt; these two are routed to the Messaging Unit Error interrupt input. The generation of interrupts recorded in the Inbound Interrupt Status Register may be masked by setting the corresponding bit in the Inbound Interrupt Mask Register. Some of the bits in this register are Read Only. For those bits, the interrupt must be cleared through another register.

Table 268. Inbound Interrupt Status Register - IISR

MU/PCI Base Address Offset IISR: 0024H		
internal bus address offset IISR: 4024H		
Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Coordinated Reset Interrupt - This bit is set when the Coordinated Reset bit (bit 1) is set in the Inbound Reset Control and Status Register.
30	0 ₂	Selective Reset Interrupt - This bit is set when the Selective Reset bit (bit 0) is set in the Inbound Reset Control and Status Register.
29	0 ₂	MU MSI-X Table Write Interrupt - This bit is set when a write occurs to any of the MU MSI-X Table entry. When set, this bit results in the assertion of the MU MSI-X Table Write Interrupt.
28:07	0000000H	Reserved
06	0 ₂	Index Register Interrupt - This bit is set by the MU hardware when an Index Register has been written after a Host I/O Interface transaction.
05	0 ₂	Outbound Free Queue Full Interrupt - This bit is set when the Outbound Free Head Pointer becomes equal to the Tail Pointer and the queue is full. An Error interrupt is generated for this condition.
04	0 ₂	Inbound Post Queue Interrupt - This bit is set by the MU hardware when the Inbound Post Queue has been written. Once cleared, an interrupt is NOT generated when the head and tail pointers remain unequal (i.e. queue status is Not Empty). Therefore, when software leaves any unprocessed messages in the post queue when the interrupt is cleared, software must retain the information that the Inbound Post queue status is not empty. Note: This interrupt is provided with dedicated support in the 81341 and 81342's Interrupt Controller.
03	0 ₂	Error Doorbell Interrupt - This bit is set when the Error Interrupt of the Inbound Doorbell Register is set. To clear this bit (and the interrupt), the Error Interrupt bit of the Inbound Doorbell Register must be clear.
02	0 ₂	Inbound Doorbell Interrupt - This bit is set when at least one Normal Interrupt bit in the Inbound Doorbell Register is set. To clear this bit (and the interrupt), the Normal Interrupt bits in the Inbound Doorbell Register must all be clear.
01	0 ₂	Inbound Message 1 Interrupt - This bit is set by the MU hardware when the Inbound Message 1 Register has been written.
00	0 ₂	Inbound Message 0 Interrupt - This bit is set by the MU hardware when the Inbound Message 0 Register has been written.

4.9.5 Inbound Interrupt Mask Register - IIMR

The Inbound Interrupt Mask Register (IIMR) provides the ability to mask Intel XScale® processor interrupts generated by the Messaging Unit. Each bit in the Mask register corresponds to an interrupt bit in the Inbound Interrupt Status Register.

Setting or clearing bits in this register does not affect the Inbound Interrupt Status Register. They only affect the generation of the Intel XScale® processor interrupt.

Table 269. Inbound Interrupt Mask Register - IIMR

MU/PCI Base Address Offset internal bus address offset IIMR: 0028H IIMR: 4028H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Coordinated Reset Interrupt Mask - When set, this bit masks the interrupt generated by the Coordinated Reset bit (bit 1) in the Inbound Reset Control and Status Register is set.
30	0 ₂	Selective Reset Interrupt Mask - When set, this bit masks the interrupt generated by the Selective Reset bit (bit 0) in the Inbound Reset Control and Status Register is set.
29	0 ₂	MU MSI-X Table Write Interrupt Mask - Controls the setting of bit 29 of the "Inbound Interrupt Status Register - IISR" and generation of the MU MSI-X Table Write Interrupt when a write occurs to any of the MU MSI-X Table entry. 0 = Not Masked 1 = Masked
28:07	000000H	Reserved
06	0 ₂	Index Register Interrupt Mask - When set, this bit masks the interrupt generated by the MU hardware when an Index Register has been written after a Host I/O Interface transaction.
05	0 ₂	Outbound Free Queue Full Interrupt Mask - When set, this bit masks the Error interrupt generated when the Outbound Free Head Pointer becomes equal to the Tail Pointer and the queue is full.
04	0 ₂	Inbound Post Queue Interrupt Mask - When set, this bit masks the interrupt generated by the MU hardware when the Inbound Post Queue has been written.
03	0 ₂	Error Doorbell Interrupt Mask - When set, this bit masks the Error Interrupt when the Error Interrupt bit of the Inbound Doorbell Register is set.
02	0 ₂	Inbound Doorbell Interrupt Mask - When set, this bit masks the interrupt generated when at least one Normal Interrupt bit in the Inbound Doorbell Register is set.
01	0 ₂	Inbound Message 1 Interrupt Mask - When set, this bit masks the Inbound Message 1 Interrupt generated by a write to the Inbound Message 1 Register.
00	0 ₂	Inbound Message 0 Interrupt Mask - When set, this bit masks the Inbound Message 0 Interrupt generated by a write to the Inbound Message 0 Register.



4.9.6 Outbound Doorbell Register - ODR

The Outbound Doorbell Register (ODR) allows software interrupt generation. It allows the Intel XScale® processor to generate Host I/O Interface interrupts to the host processor by writing to the Software Interrupt bits or to a specific Host I/O Interface interrupt bit. The generation of Host I/O Interface interrupts through the Outbound Doorbell Register may be masked by setting the Outbound Doorbell Interrupt Mask bit in the Outbound Interrupt Mask Register.

The Software Interrupt bits in this register can only be set by the Intel XScale® processor and can only be cleared by an external Host I/O Interface agent.

Table 270. Outbound Doorbell Register - ODR

MU/PCI Base Address Offset ODR: 002CH	internal bus address offset ODR: 402CH	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description	
31	0 ₂	PCI Interrupt D - When set, this bit causes the P_INTD# signal to be asserted or a Message-signaled Interrupt is generated (when enabled). When this bit is cleared, the P_INTD# signal is deasserted.	
30	0 ₂	PCI Interrupt C - When set, this bit causes the P_INTC# signal to be asserted or a Message-signaled Interrupt is generated (when enabled). When this bit is cleared, the P_INTC# signal is deasserted.	
29	0 ₂	PCI Interrupt B - When set, this bit causes the P_INTB# signal to be asserted or a Message-signaled Interrupt is generated (when enabled). When this bit is cleared, the P_INTB# signal is deasserted.	
28	0 ₂	PCI Interrupt A - When set, this bit causes the P_INTA# signal to be asserted or a Message-signaled Interrupt is generated (when enabled). When this bit is cleared, the P_INTA# signal is deasserted.	
27:00	000000H	Software Interrupt - When any bit is set, generate a PCI interrupt or a Message-signaled interrupt is generated (when enabled). The PCI interrupt pin used is determined by the ATU Interrupt Pin Register. When all bits are clear, do not generate a PCI interrupt.	

4.9.7 Outbound Interrupt Status Register - OISR

The Outbound Interrupt Status Register (OISR) contains hardware interrupt status. It records the status of Host I/O Interface interrupts generated by the Message Registers, Doorbell Registers, and the Circular Queues. The generation of Host I/O Interface interrupts recorded in the Outbound Interrupt Status Register may be masked by setting the corresponding bit in the Outbound Interrupt Mask Register. Some of the bits in this register are Read Only. For those bits, the interrupt must be cleared through another register.

Table 271. Outbound Interrupt Status Register - OISR

MU/PCI Base Address Offset
OISR: 0030H

internal bus address offset
OISR: 4030H

Attribute Legend:
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

Bit	Default	Description
31	0 ₂	Firmware Interrupt Pending - This bit is set when the Firmware Interrupt bit is set in the Outbound Reset Control and Status Register. To clear this bit (and the interrupt), the Firmware Interrupt bit must be cleared in the Outbound Reset Control and Status Register.
30:08	000000H 000 ₂	Reserved
07	0 ₂	PCI Interrupt D - This bit is set when the PCI Interrupt D bit is set in the Outbound Doorbell Register. To clear this bit (and the interrupt), the PCI Interrupt D bit must be cleared.
06	0 ₂	PCI Interrupt C - This bit is set when the PCI Interrupt C bit is set in the Outbound Doorbell Register. To clear this bit (and the interrupt), the PCI Interrupt C bit must be cleared.
05	0 ₂	PCI Interrupt B - This bit is set when the PCI Interrupt B bit is set in the Outbound Doorbell Register. To clear this bit (and the interrupt), the PCI Interrupt B bit must be cleared.
04	0 ₂	PCI Interrupt A - This bit is set when the PCI Interrupt A bit is set in the Outbound Doorbell Register. To clear this bit (and the interrupt), the PCI Interrupt A bit must be cleared.
03	0 ₂	Outbound Post Queue Interrupt - This bit is set when data in the prefetch buffer is valid. This bit is cleared when any prefetch data has been read from the Outbound Queue Port.
02	0 ₂	Outbound Doorbell Interrupt - This bit is set when at least one Software Interrupt bit in the Outbound Doorbell Register is set. To clear this bit (and the interrupt), the Software Interrupt bits in the Outbound Doorbell Register must all be clear.
01	0 ₂	Outbound Message 1 Interrupt - This bit is set by the MU when the Outbound Message 1 Register is written. Clearing this bit clears the interrupt.
00	0 ₂	Outbound Message 0 Interrupt - This bit is set by the MU when the Outbound Message 0 Register is written. Clearing this bit clears the interrupt.



4.9.8 Outbound Interrupt Mask Register - OIMR

The Outbound Interrupt Mask Register (OIMR) provides the ability to mask outbound Host I/O Interface interrupts generated by the Messaging Unit. Each bit in the mask register corresponds to a hardware interrupt bit in the Outbound Interrupt Status Register. When the bit is set, the Host I/O Interface interrupt is not generated. When the bit is clear, the interrupt is allowed to be generated.

Setting or clearing bits in this register does not affect the Outbound Interrupt Status Register. They only affect the generation of the Host I/O Interface interrupt.

Table 272. Outbound Interrupt Mask Register - OIMR

MU/PCI Base Address Offset OIMR: 0034H	internal bus address offset OIMR: 4034H	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	Firmware Interrupt Mask - When set, this bit masks the Firmware Reset interrupt signal when the Firmware interrupt bit in the Outbound Reset Control and Status Register is set.
30:05	000000H	Reserved
07	0 ₂	PCI Interrupt D Mask - When set, this bit masks the PCI Interrupt D signal when the PCI Interrupt D bit in the in the Outbound Doorbell Register is set.
06	0 ₂	PCI Interrupt C Mask - When set, this bit masks the PCI Interrupt C signal when the PCI Interrupt C bit in the in the Outbound Doorbell Register is set.
05	0 ₂	PCI Interrupt B Mask - When set, this bit masks the PCI Interrupt B signal when the PCI Interrupt B bit in the in the Outbound Doorbell Register is set.
04	0 ₂	PCI Interrupt A Mask - When set, this bit masks the PCI Interrupt A signal when the PCI Interrupt A bit in the in the Outbound Doorbell Register is set.
03	0 ₂	Outbound Post Queue Interrupt Mask - When set, this bit masks the PCI interrupt generated when data in the prefetch buffer is valid.
02	0 ₂	Outbound Doorbell Interrupt Mask - When set, this bit masks the Software Interrupt generated by the Outbound Doorbell Register.
01	0 ₂	Outbound Message 1 Interrupt Mask - When set, this bit masks the Outbound Message 1 Interrupt generated by a write to the Outbound Message 1 Register.
00	0 ₂	Outbound Message 0 Interrupt Mask- When set, this bit masks the Outbound Message 0 Interrupt generated by a write to the Outbound Message 0 Register.

4.9.9 Inbound Reset Control and Status Register - IRCSR

The Inbound Reset Control and Status Register (IRCSR) provides the ability for the Host processor to request a Selective Reset or a Coordinated Reset. A selective reset is used to perform a soft reset. A selective reset is requested by the host processor setting the Selective Reset bit of the IRCSR, which causes an Intel XScale® processor interrupt.

A coordinated reset is used when supporting multiple PCI functions. In a multi-function scenario, before the Host driver can issue a hardware reset via one of the PCI functions, all the host drivers running must be quiesced. A coordinated reset is requested by the host processor setting the Coordinated Reset bit in the IRCSR, which causes an Intel XScale® processor interrupt. After all the host drivers have been quiesced, the host driver that initiated the coordinated reset would request a hardware reset of the 81341 and 81342. When the host driver sets both the Selective Reset and Coordinated Reset bits simultaneously, an internal bus reset is initiated.

Note: An internal bus reset event that is caused by setting both the CR and SR bits of the IRCSR is indicated in the Reset Cause and Status Register (RCSR). For example, after an internal bus reset the cause of the internal bus reset can be identified by reading the RCSR. Refer to the [Chapter 6.0, “System Controller \(SC\) and Internal Bus Bridge”](#) for detailed descriptions of the RCSR register.

Table 273. Inbound Reset Control and Status Register - IRCSR

MU/PCI Base Address Offset OIMR: 0038H		internal bus address offset OIMR: 4038H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set		RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description					
31:02	00000000H	Reserved.					
01	0 ₂	Coordinated Reset (CR) - This bit is valid when the 81341 and 81342 implements Multi-Function Reset protocol, (MF=1, bit 1 in the ORCSR). A Host driver writes a one into this bit to request a Coordinated Reset. Writing a 1 to the CR bit would generate an interrupt to the Intel XScale® processor. However, writing 1's to the CR bit and the SR bit simultaneously would instead generate an internal bus reset. In a Coordinated Reset (only CR bit set), each function is notified and acknowledges the internal bus reset before it actually occurs. In an internal bus reset, the reset proceeds immediately without any notification to the any of the other Hosts.					
00	0 ₂	Selective Reset (SR) - The Host driver sets this bit in order to request a firmware state and structures re-initialization. Setting this bit generates an interrupt to the Intel XScale® processor. However, writing 1's to the CR bit and the SR bit simultaneously would instead generate an internal bus reset. Setting the SR bit by itself does not generate an internal bus reset. Therefore, the Intel XScale® processor must be operational for the reset to be processed. Setting this bit has no impact on any other functions on 81341 and 81342.					



4.9.10 Outbound Reset Control and Status Register - ORCSR

The Outbound Reset Control and Status Register (ORCSR) provides the ability for the I/O-processor to coordinate a hardware reset with the Host processor when multi-function is being used. In a multi-function scenario, before the Host driver can issue a hardware reset via one of the functions, all the host drivers running must be quiesced.

Note: GRO and RM bits of the ORCSR are saved in the Reset Cause Status Register (RCSR) when both CR and SR bits of the IRCSR are set to cause an internal bus reset. The Reset Cause Status Register implements sticky bits. The initiator of the internal bus reset is also indicated in the RCSR. Refer to the [Exception Initiator and Boot Sequence Chapter](#) for detailed descriptions of the RCSR register.

Table 274. Outbound Reset Control and Status Register - ORCSR

MU/PCI Base Address Offset OIMR: 003CH	internal bus address offset OIMR: 403CH	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	Firmware Interrupt - This bit is set by the firmware to initiate an interrupt to the Host processor and this bit is used during a coordinated reset handshaking sequence. Firmware sets this bit. The host clears this bit by writing a 1.
30:04	00000H	Reserved.
03	0 ₂	Global Reset Outstanding (GRO) - This bit is only used when the 81341 and 81342 implements Multi-Function Reset protocol (MF=1, bit 1 in the ORCSR). When the Host Processor request a coordinated reset in one of the Function Interfaces, and the 81341 and 81342 implements Multi-function Reset protocol, the 81341 and 81342 firmware sets the GRO bit on all function interfaces. When Multi-Function Reset protocol is not implemented, this bit is always zero. Note: The state of this bit is saved in the Reset Cause Status Register (RCSR) when an internal bus reset occurs. Refer to the Exception Initiator and Boot Sequence Chapter for detailed descriptions of the RCSR register.
02	0 ₂	Reset Coordination Complete (RCC) - This bit is used during a Coordinated Reset to indicate to the Reset Master that all the other Host Drivers have acknowledged the pending reset and that the Reset Master may proceed with the reset. The Coordinated Reset is only available on 81341 and 81342 that implement the Multi-function Feature (MF=1, bit 1 in the ORCSR). This bit is only set by the Core when GRO and RM are both set.
01	0 ₂	Multi-Function (MF) - The Intel XScale® processor sets this bit to indicate that the 81341 and 81342 supports multiple PCI function features such as multi-function reset.
00	0 ₂	Reset Master (RM) - This bit is set by firmware when one of the Host drivers requests a Coordinated Reset and this Interface takes on the Reset Master role during the Coordinated Reset. This feature is only available when the MF bit is set, bit 1 in the ORCSR. The Host driver requests a Coordinated Reset by writing a 1 into the CR bit, bit 1 of the IRCSR. Note: The state of this bit is saved in the Reset Cause Status Register (RCSR) when an internal bus reset occurs. Refer to the Exception Initiator and Boot Sequence Chapter for detailed descriptions of the RCSR register.

4.9.11 MSI Inbound Message Register — MIMR

The MSI Inbound Message (MIMR) is a 16-bit data register that can be used to receive inbound MSI interrupt (Message-Signaled Interrupt) from external PCI devices. When operating as a Root Complex (ATU-E) or Central Resource (ATU-X) device, an external PCI device can signal an interrupt by writing the MSI Inbound Message Register. The MU interprets the data bits of MIMR as follows:

- Bit[15] of MIMR is used to select the Intel XScale® processors to be the target of the MSI interrupt. For example, the MSI interrupt can be targeted to either Intel XScale® processor 0 (coreID = 0) or Intel XScale® processor 1 (coreID = 1). Refer to the EI_BS chapter for more details on coreID assignments.
- Bits[14:07] of MIMR are reserved.

Bits[06:00] of MIMR are treated as a vector field which is one-hot decoded and the bits are posted in Intel XScale® processor co-processor registers. Refer to [Section 4.9.42, “Inbound MSI Interrupt Pending Register x — IMIPRx”](#) for more details on IMIPR[0:3].

Table 275. MSI Inbound Message Register - MIMR

MU/PCI Base Address Offset MIMR: 0048H	internal bus address offset MIMR: 4048H	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:16	0000H	Reserved
15	0 ₂	Core Select Bit - This bit is used to select the Intel XScale® processor which is the target of the MSI Interrupt. 0 = Intel XScale® processor0 (coreID == 0) 1 = Intel XScale® processor1 (coreID == 1)
14:07	00H	Reserved
06:00	0000000 ₂	Interrupt Vector - This bit field is used to generate a one-hot bit which is posted in Intel XScale® processor co-processor registers (Section 4.9.42, “Inbound MSI Interrupt Pending Register x — IMIPRx”). Each interrupt vector value is associated with one bit in the co-processor registers. For example, interrupt vector value == 0 is associated with bit 0 of IMIPR0, interrupt vector value == 32 is associated with bit 0 of IMIPR1, interrupt vector value == 64 is associated with bit 0 of IMIPR2, and so on. Refer to Section 4.9.42, “Inbound MSI Interrupt Pending Register x — IMIPRx” for more details on IMIPR[0:3].



4.9.12 MU Configuration Register - MUCR

The MU Configuration Register (MUCR) contains the Circular Queue Enable bit and the size of one Circular Queue. The Circular Queue Enable bit enables or disables the Circular Queues. The Circular Queues are disabled at reset to allow the software to initialize the head and tail pointer registers before any PCI accesses to the Queue Ports. Each Circular Queue may range from 4 K entries (16 Kbytes) to 64 K entries (256 Kbytes) and there are four Circular Queues.

This register also contains the upper four bits of the 36-bit QBR address. Local memory is 36-bit addressable.

Table 276. MU Configuration Register - MUCR

MUCR	internal bus address offset 4050H	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:20	000H	Reserved
19:16	0H	Upper Queue Base Address - These four bits contain the upper 4-bit address of the 36-bit QBR. Local memory is 36-bit addressable.
15:06	000000H	Reserved
05:01	00001 ₂	Circular Queue Size - This field determines the size of each Circular Queue. All four queues are the same size. 00001₂ - 4K Entries (16 Kbytes) 00010₂ - 8K Entries (32 Kbytes) 00100₂ - 16K Entries (64 Kbytes) 01000₂ - 32K Entries (128 Kbytes) 10000₂ - 64K Entries (256 Kbytes)
00	0 ₂	Circular Queue Enable - This bit enables or disables the Circular Queues. When clear the Circular Queues are disabled, however the MU accepts Host I/O Interface accesses to the Circular Queue Ports but ignores the data for Writes and return FFFF.FFFFH for Reads. Interrupts are not generated to the core when disabled. When set, the Circular Queues are fully enabled.

4.9.13 Queue Base Address Register - QBAR

The Queue Base Address Register (QBAR) contains the lower 32-bit of the 36-bit local memory address of the Circular Queues. The base address is required to be located on a 1 Mbyte address boundary. The upper four-bits of the QBAR is located in the MU Configuration Register (MUCR).

All Circular Queue head and tail pointers are based on the QBAR. When the head and tail pointer registers are read, the Queue Base Address is returned in the upper 12 bits. Writing to the upper 12 bits of the head and tail pointer registers does not affect the Queue Base Address or Queue Base Address Register.

Warning: The QBAR must designate a range allocated to the 81341 and 81342's DDR SDRAM interface ([Chapter 7.0, "DDR SDRAM Memory Controller"](#)).

Table 277. Queue Base Address Register - QBAR

312824201612840 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na internal bus address offset 4054H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
QBAR		
Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:00	00000H	Reserved



4.9.14 Inbound Free Head Pointer Register - IFHPR

Inbound Free Head Pointer Register (IFHPR) contains local memory offset from the Queue Base Address of the head pointer for the Inbound Free Queue. The Head Pointer must be aligned on a word address boundary. When read, the Queue Base Address is provided in the register upper 12 bits. Writes to the upper 12 bits of the register are ignored. This register is maintained by software.

Table 278. Inbound Free Head Pointer Register - IFHPR

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00 ₂	Inbound Free Head Pointer - Local memory offset of the head pointer for the Inbound Free Queue.
01:00	00 ₂	Reserved

4.9.15 Inbound Free Tail Pointer Register - IFTPR

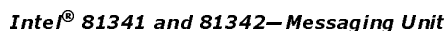
The Inbound Free Tail Pointer Register (IFTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Inbound Free Queue. The Tail Pointer must be aligned on a word address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 279. Inbound Free Tail Pointer Register - IFTPR

IFTPR

internal bus address offset 4064H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 RC = Read Clear
 RO = Read Only
 PR = Preserved
 RS = Read/Set
 NA = Not Accessible



Inbound Post Head Pointer Register - IPHPR

Inbound Post Tail Pointer Register - IPTPR



4.9.18 Outbound Free Head Pointer Register - OFHPR

The Outbound Free Head Pointer Register (OFHPR) contains the local memory offset from the Queue Base Address of the head pointer for the Outbound Free Queue. The Head Pointer must be aligned on a word address boundary. This register is maintained by software. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 282. Outbound Free Head Pointer Register - OFHPR

OFHPR 4070H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible

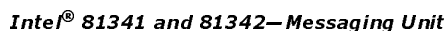
Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00 ₂	Outbound Free Head Pointer - Local memory offset of the head pointer for the Outbound Free Queue.
01:00	00 ₂	Reserved

4.9.19 Outbound Free Tail Pointer Register - OFTPR

The Outbound Free Tail Pointer Register (OFTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Outbound Free Queue. The Tail Pointer must be aligned on a word address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 283. Outbound Free Tail Pointer Register - OFTPR

Diagram illustrating the OFTPR (Outbound Free Tail Pointer Register) structure. The register is 32 bits wide, divided into two 16-bit sections: IOP Attributes (bits 31:16) and PCI Attributes (bits 15:0). The IOP Attributes section contains 16 bits, with the last 4 bits (bits 12:9) reserved (RV). The PCI Attributes section contains 16 bits, with the last 4 bits (bits 12:9) reserved (RV). The diagram shows the internal bus address offset 4074H. The Attribute Legend indicates: RW = Read/Write, RV = Reserved, RC = Read Clear, RO = Read Only, PR = Preserved, RS = Read/Set, NA = Not Accessible.



Outbound Post Head Pointer Register - OPHPR

Outbound Post Tail Pointer Register - OPTPR



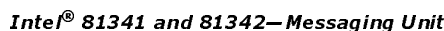
4.9.22 Index Address Register - IAR

The Index Address Register (IAR) contains the offset of the least recently accessed Index Register. It is written by the MU when the Index Registers are written by a Host I/O Interface agent. The register is not updated until the Index Interrupt bit in the Inbound Interrupt Status Register is cleared.

The local memory address of the Index Register least recently accessed is computed by adding the Index Address Register to the Inbound ATU Translate Value Register.

Table 286. Index Address Register - IAR

IAR internal bus address offset 4080H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:12	000000H	Reserved
11:02	00H 00 ₂	Index Address - is the local memory offset of the Index Register written (050H to FFCH)
01:00	00 ₂	Reserved

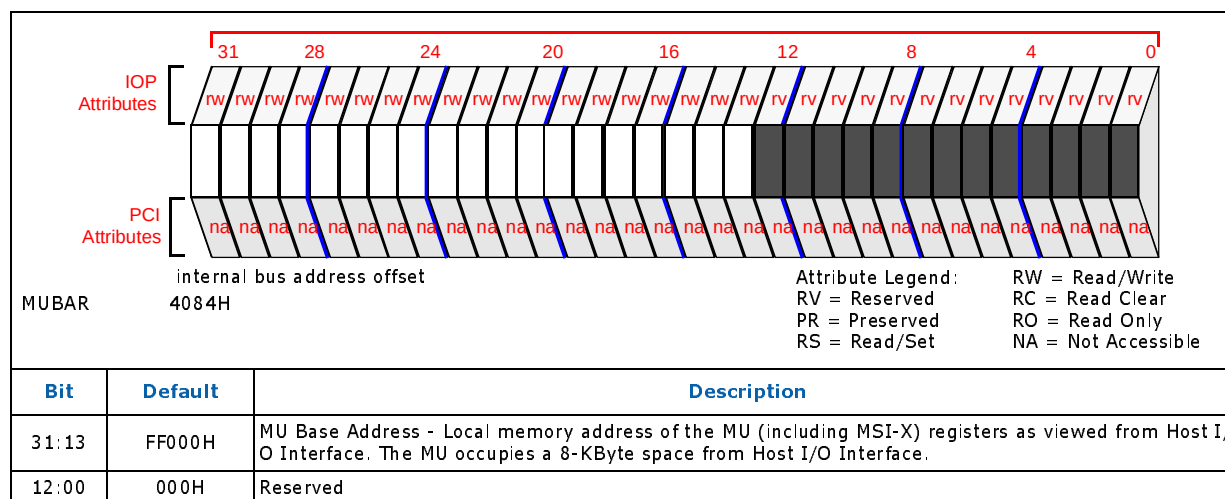


MU Base Address Register - MUBAR

MU Base Address Register (MUBAR) contains lower 32-bit of the 36-bit local memory base address of the MU address space as depicted in [Figure 41](#). For example, the MU address space as viewed from Host I/O Interface. The MU base address is required to be located on an 8-KByte boundary. The upper four-bits of the MU Base Address are located in the MU Upper Address Register (MUBAR). Refer to [Section 4.9.24, “MU Upper Base Address Register - MUUBAR”](#). The MU always claim the entire 8 KBytes of the internal bus address space relative to the MU Base Address Registers. The 8 KBytes include the MU Registers, Index Registers, and MSI-X Data Structures. When the 8-KByte MU Base Address Registers are programmed to overlap onto the DDR MCU Memory space, the entire overlapped 8-KByte space is claimed by the MU - and not by the DDR MCU. Overlapping the MU space onto the DDR MCU space is required when the Index registers are required, because the Index registers are implemented using the DDR Memory.

Note: The default values of MUBAR/MUBAR are programmed to match the default values programmed in the Inbound ATU Translate Value Register 0 - IATVR0/Inbound ATU Upper Translate Value Register 0 - IAUTVR0. This allows the MU registers to be mapped in the first 8-KByte of the PCI Window 0 Address space.

Table 287. MU Base Address Register - MUBAR





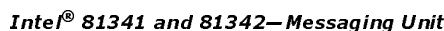
4.9.24 MU Upper Base Address Register - MUUBAR

The MU Upper Base Address Register (MUUBAR) contains the upper 4-bit of the 36-bit local memory base address of the MU address space as depicted in Figure 41. For example, the MU address space as viewed from Host I/O Interface interface. The MU base address is required to be located on a 8-KByte boundary. The lower 32-bits of the MU Base Address are located in the MU Upper Address Register (MUBAR). Refer to Section 4.9.23, “MU Base Address Register - MUBAR” on page 444.

Note: The default values of MUBAR/MUUBAR are programmed to match the default values programmed in the Inbound ATU Translate Value Register 0 - IATVR0/Inbound ATU Upper Translate Value Register 0 - IAUTVR0. This allows the MU registers to be mapped in the first 8-KByte of the PCI Window 0 Address space.

Table 288. MU Upper Base Address Register - MUUBAR

Bit	Default	Description
31:04	0000000H	Reserved
03:00	0H	MU Upper Base Address - Local memory address of the MU (including MSI-X) registers as viewed from Host I/O Interface. The MU occupies a 8-KByte space from Host I/O Interface.



The MU MSI-X Table Message Address Register contains the lower 30 bits of the MSI-X message address. An entry in the MSI-X Table is made up of four DWORDs.

Note: The M MT MAR[0:7] registers are not reset with an internal bus reset.

Table 289. MU MSI-X Table Message Address Registers - M_MT_MAR [0:7]

The diagram illustrates the structure of the MSI-X Message Address field, which is 32 bits wide. It is divided into two 16-bit halves: the top half for IOP Attributes and the bottom half for PCI Attributes. Each half is further divided into four 4-bit segments, labeled 31, 28, 24, 20, 16, 12, 8, and 4. The bottom half segment 0 is shaded gray. The bit patterns for each segment are: RW (Read/Write), RV (Reserved), RC (Read Clear), RO (Read Only), and RS (Read/Set).

MU/PCI Base Address Offset	internal bus address offset	Attribute Legend:	
M_MT_MAR0 1000H	M_MT_MAR0 5000H	RV = Reserved	RW = Read/Write
M_MT_MAR1 1010H	M_MT_MAR1 5010H	PR = Preserved	RC = Read Clear
M_MT_MAR2 1020H	M_MT_MAR2 5020H	RS = Read/Set	RO = Read Only
M_MT_MAR3 1030H	M_MT_MAR3 5030H		NA = Not Accessible
M_MT_MAR4 1040H	M_MT_MAR4 5040H		
M_MT_MAR5 1050H	M_MT_MAR5 5050H		
M_MT_MAR6 1060H	M_MT_MAR6 5060H		
M_MT_MAR7 1070H	M_MT_MAR7 5070H		

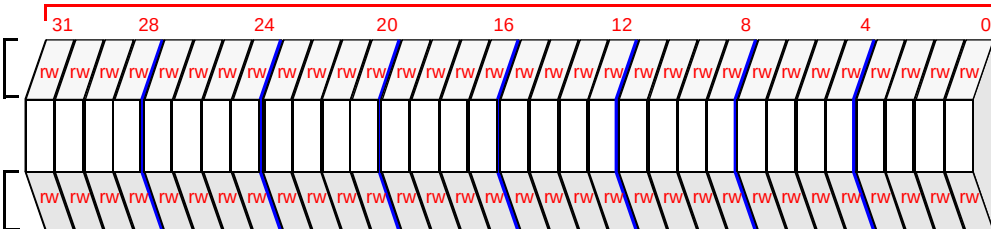


4.9.26 MU MSI-X Table Message Upper Address Registers - M_MT_MUAR[0:7]

The MU MSI-X Table Message Upper Address Register contains the upper 32 bits of the MSI-X message address. An entry in the MSI-X Table is made up of four DWORDs.

Note: The M_MT_MUAR[0:7] registers are not reset with an internal bus reset.

Table 290. MU MSI-X Table Message Upper Address Registers - M_MT_MUAR [0:7]

																																							
		MU/PCI Base Address Offset																internal bus address offset																Attribute Legend:		RW = Read/Write			
																																		RV = Reserved		RC = Read Clear			
																																		PR = Preserved		RO = Read Only			
																																		RS = Read/Set		NA = Not Accessible			
M_MT_MAR0	10 04H																	M_MT_MAR0	50 04H																				
M_MT_MAR1	10 14H																	M_MT_MAR1	50 14H																				
M_MT_MAR2	10 24H																	M_MT_MAR2	50 24H																				
M_MT_MAR3	10 34H																	M_MT_MAR3	50 34H																				
M_MT_MAR4	10 44H																	M_MT_MAR4	50 44H																				
M_MT_MAR5	10 54H																	M_MT_MAR5	50 54H																				
M_MT_MAR6	10 64H																	M_MT_MAR6	50 64H																				
M_MT_MAR7	10 74H																	M_MT_MAR7	50 74H																				

Bit	Default	Description
31:00	0000 0000 H	Message Upper Address: This field contains the upper 32 bits of the MSI-X Message Address.



4.9.27 MU MSI-X Table Message Data Registers - M_MT_MDR[0:7]

The MU MSI-X Table Message Data Register contains the message data of the MSI-X message. An entry in the MSI-X Table is made up of four DWORDs.

Note: The M_MT_MDR[0:7] registers are not reset with an internal bus reset.

Table 291. MU MSI-X Table Message Upper Address Registers - M_MT_MUAR [0:7]

		MU/PCI Base Address Offset internal bus address offset Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible	
M_MT_MAR0	1008H	M_MT_MAR0	5008H
M_MT_MAR1	1018H	M_MT_MAR1	5018H
M_MT_MAR2	1028H	M_MT_MAR2	5028H
M_MT_MAR3	1038H	M_MT_MAR3	5038H
M_MT_MAR4	1048H	M_MT_MAR4	5048H
M_MT_MAR5	1058H	M_MT_MAR5	5058H
M_MT_MAR6	1068H	M_MT_MAR6	5068H
M_MT_MAR7	1078H	M_MT_MAR7	5078H

Bit	Default	Description
31:00	0000 0000H	Message Data: This field contains the message data for this Table entry.

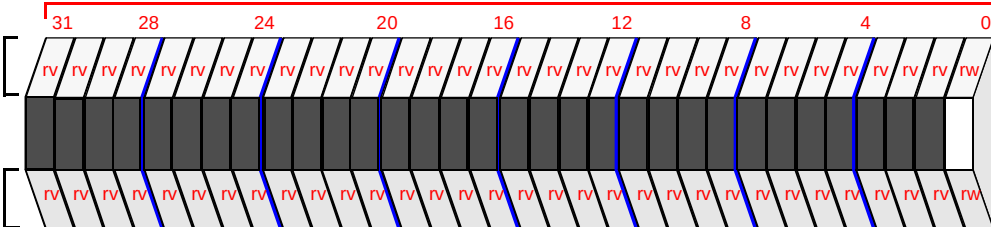


4.9.28 MU MSI-X Table Message Vector Control Registers - M_MT_MVCR[0:7]

The MU MSI-X Table Message Vector Control Register contains the mask bit for this entry in the MSI-X Table. An entry in the MSI-X Table is made up of four DWORDs.

Note: The M_MT_MVCR[0:7] registers are not reset with an internal bus reset.

Table 292. MU MSI-X Table Message Vector Control Registers - M_MT_MVCR [0:7]

																																								
		<table><tr><td>MU/PCI Base Address</td><td>internal bus address</td><td rowspan="2">Attribute Legend:</td><td rowspan="2">RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible</td></tr><tr><td>Offset</td><td>offset</td></tr><tr><td>M_MT_MAR0</td><td>M_MT_MAR0</td><td></td><td></td></tr><tr><td>M_MT_MAR1</td><td>M_MT_MAR1</td><td></td><td></td></tr><tr><td>M_MT_MAR2</td><td>M_MT_MAR2</td><td></td><td></td></tr><tr><td>M_MT_MAR3</td><td>M_MT_MAR3</td><td></td><td></td></tr><tr><td>M_MT_MAR4</td><td>M_MT_MAR4</td><td></td><td></td></tr><tr><td>M_MT_MAR5</td><td>M_MT_MAR5</td><td></td><td></td></tr><tr><td>M_MT_MAR6</td><td>M_MT_MAR6</td><td></td><td></td></tr><tr><td>M_MT_MAR7</td><td>M_MT_MAR7</td><td></td><td></td></tr></table>	MU/PCI Base Address	internal bus address	Attribute Legend:	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	Offset	offset	M_MT_MAR0	M_MT_MAR0			M_MT_MAR1	M_MT_MAR1			M_MT_MAR2	M_MT_MAR2			M_MT_MAR3	M_MT_MAR3			M_MT_MAR4	M_MT_MAR4			M_MT_MAR5	M_MT_MAR5			M_MT_MAR6	M_MT_MAR6			M_MT_MAR7	M_MT_MAR7		
MU/PCI Base Address	internal bus address	Attribute Legend:	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																																					
Offset	offset																																							
M_MT_MAR0	M_MT_MAR0																																							
M_MT_MAR1	M_MT_MAR1																																							
M_MT_MAR2	M_MT_MAR2																																							
M_MT_MAR3	M_MT_MAR3																																							
M_MT_MAR4	M_MT_MAR4																																							
M_MT_MAR5	M_MT_MAR5																																							
M_MT_MAR6	M_MT_MAR6																																							
M_MT_MAR7	M_MT_MAR7																																							
Bit	Default	Description																																						
31:01	0000 0000H	Reserved.																																						
00	1 ₂	Message Vector Control: This bit when set, prohibits the sending an MSI-X message using this entry.																																						



4.9.29 MU MSI-X Pending Bits Array Register - M_MPBAR

The MU MSI-X Pending Bits Array Register contains the pending bits for the eight MU interrupt sources. When an entry in the MSI-X table is masked in the Vector Control Register, the software may service that interrupt request by polling the pending bit.

Note: The M_MPBAR register is not reset with an internal bus reset.

Table 293. MU MSI-X Pending Bits Array Register - M_MPBAR

Bit	Default	Description
31:08	0000 000H	Reserved.
07:00	00000000 ₂	Pending Bits Array: Any bit that is set indicates that the associated MSI-X message is scheduled to be sent. When an associated entry in the MSI-X table is masked using the mask bit in the Vector Control Register, software can service the interrupt based on polling the associated pending bit.

4.9.30 MSI Capability Identifier Register - Cap_ID

The MSI Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.2. This register in the PCI Interface Extended Capability header identifies the type of Extended Capability contained in that header. The value of 05H in this field identifies the function as message signaled interrupt capable.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 294. MSI Capability Identifier Register - MSI_Cap_ID

Bit	Default	Description
07:00	05H	MSI_Cap_Id - This field with its 05H value identifies this item in the linked list of Extended Capability Headers as being the message signaled interrupt capability item.



4.9.31 MSI Next Item Pointer Register - MSI_Next_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.2. This register describes the location of the next item in the function capability list. For the 81341 and 81342 that is the PCI-X capability header at offset E0H.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 295. MSI Next Item Pointer Register - MSI_Next_Ptr

Internal Bus Address Offset 0A1H PCI Configuration Offset A1H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	D0H	MSI_Next_Pointer - This field provides an offset into the function configuration space pointing to the next item in the function capability list



4.9.32 Message Control Register - Message_Control

The Message Control Register provides system software control over MSI. After reset, MSI is disabled. System software is permitted to modify the Message Control register's read/write bits and fields while a device driver is not permitted to modify them.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 296. Message Control Register - Message_Control

IOP Attributes PCI Attributes		
Internal Bus Address Offset 0A2H	PCI Configuration Offset A2H	Attribute Legend: RW = Read/Write RC = Read Clear RO = Read Only RS = Read/Set NA = Not Accessible
Bit	Default	Description
15:8	00H	Reserved
7	1 ₂	64-bit Address Support - This field is set to 1 ₂ indicating that the 81341 and 81342 is capable of generating a 64-bit message address.
6:4	000 ₂	Multiple Message Enable - System software writes to this field to indicate the number of messages allocated to the 81341 and 81342. While, the 81341 and 81342 requests two messages, it is possible that system software only allocates one message. The device hardware is designed to handle both cases.
3:1	001 ₂	Multiple Message Capable - This field is set to 001 ₂ indicating that the 81341 and 81342 can issue up to two unique interrupt messages.
0	0 ₂	MSI Enable - Setting this bit enables the 81341 and 81342's MSI functionality and disables the use of the P_INTA# interrupt output for 81341 and 81342 interrupts.



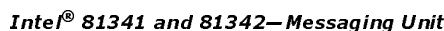
4.9.33 Message Address Register - Message_Address

The Message address register specifies the DWORD aligned address for the MSI memory write transaction. The value is set by system software during initialization.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 297. Message Address Register - Message_Address

		Internal Bus Address Offset 0A4H		PCI Configuration Offset A4H		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description					
31:2	00000000H	Message Address - DWORD aligned Message Address. This value is set by system software.					
1:0	00 ₂	Reserved.					



The Message Upper Address register is set during system initialization when system software wishes to place the MSI address location above the 4G address boundary. When this register is set to a non-zero value, the 81341 and 81342 generates a dual address cycle for the MSI write command and uses the contents of this register as the upper 32-bits of that address.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Internal Bus Address Offset 0A8H PCI Configuration Offset A8H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	Message Upper Address - Upper 32 bits of a 64-bit PCI address. This value is set by system software.



4.9.35 Message Data Register- Message_Data

The value in the Message Data Register contains the data used during an MSI write transaction. When two unique messages are enabled, one message is reserved for the Outbound Post Queue Interrupt and the other message represents all of the Outbound Doorbell and Outbound Message Interrupts. When only one message is enabled, all of these interrupts are represented by a single message. Interrupt handler software needs to read the 81341 and 81342 Outbound Interrupt Status Register to determine the cause of the interrupt when more than one source is represented by a single message.

During an MSI write data phase, the value in the Message Data Register is driven on to **AD[15:0]** while **AD[31:16]** are driven to zero. **C/BE[3:0]#** are asserted during the data phase of the memory write transaction.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 299. Message Data Register - Message_Data

Internal Bus Address Offset 0ACH PCI Configuration Offset ACH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
15:00	0000H	Message Data - System software specifies a 16-bit value to be transferred during the data phase of an MSI write transaction. During initialization, system software can allocate two messages to the 81341 and 81342 by writing "001" to the Multiple Message Enable field of the Message Control Register. In this case, the hardware shall modify bit 0 of the Message Data value before generating the MSI write transaction. The Host I/O Interface Outbound interrupts are distributed across the two messages by the setting and clearing of bit 0 of the Message Data Register's contents before the data is used for the MSI write: 0 = Outbound Post Queue Interrupt 1 = Outbound Doorbell and Outbound Message Interrupts When software leaves the Multiple Message Enable field of the Message Control Register at its' default value of "000", the 81341 and 81342 has only been allocated one message. Consequently, the value in the MSI data register is transmitted unmodified during an MSI write cycle.



4.9.36 MSI-X Capability Identifier Register - MSI-X_Cap_ID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 81341 and 81342, this is the MSI-X extended capability with an ID of 0DH as defined by the *PCI-X Protocol Addendum to the PCI Local Bus Specification*, Revision 2.0.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 300. MSI-X_Capability Identifier Register - MSI-X_Cap_ID

Internal Bus Address Offset 0B0H PCI Configuration Offset B0H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	0DH	Cap_Id - This field with its 0DH value identifies this item in the linked list of Extended Capability Headers as being the MSI-X capability registers.



4.9.37 MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 81341 and 81342, the next capability (PCI-X capability list) is located at off-set E0H.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 301. MSI-X Next Item Pointer Register - MSI-X_Next_Item_Ptr

Internal Bus Address Offset 480B1H PCI Configuration Offset B1H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07:00	A0H	Next_Item_Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list which in the 81341 and 81342 is the MSI extended capabilities header.



4.9.38 MSI-X Message Control Register - MSI-X_MCR

MSI-X Capabilities bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register is a 16-bit read-only register which provides information on the capabilities of the ATU function related to Message Signaled Interrupts.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 302. MSI-X Message Control Register - MSI-X_MCR

Internal Bus Address Offset 0B2H	PCI Configuration Offset B2 - B3H	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
15	0 ₂	MSI-X Enable: When set, the 81341 and 81342 is able to use MSI-X to request service.
14	0 ₂	Function Mask: When set, all the vectors in the MSI-X Table are globally masked, regardless of the per-vector Mask Bit states in the Vector Control Register of the MSI-X Table entries.
13:11	000 ₂	Reserved
10:00	0000000000 ₂ or 00000000111 ₂ (See description for default value)	MSI-X Table Size: This field indicates the MSI-X Table size N. This field is encoded as N-1. Up to eight messages can be generated. However, when the MSI-X Single Message Vector bit is set in the "MU MSI-X Control Register X — MMCRx" on page 461, only a single MSI-X message is generated. The value of this register field is dependent on the setting of the MSI-X Single Message Vector bit.
		MSI-X Single Message Bit Default Value
		0 00000000111 ₂
		1 00000000000 ₂



4.9.39 MSI-X Table Offset Register — MSI-X_Table_Offset

This register indicates in which PCI Memory Window the MSI-X Table is mapped. This register also provides an offset in the specified PCI Memory Window of where the MSI-X Table begins.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 303. MSI-X Table Offset Register - MSI-X_Table_Offset



4.9.40 MSI-X Pending Bit Array Offset Register - MSI-X_PBA_Offset

This register indicates in which PCI Memory Window the MSI-X PBA is mapped. This register also provides an offset in the specified PCI Memory Window of where the MSI-X PBA begins.

Note: Refer to the Peripheral Registers Chapter for the default internal bus address. This register is part of the configuration space of the Address Translation Unit that is setup as an endpoint.

Table 304. MSI-X Pending Bit Array Offset Register - MSI-X_PBA Offset

31 28 24 20 16 12 8 4 0 IOP Attributes rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rv rv rv rv rv rv rv rv rv rv rw rw rw rw PCI Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Internal Bus Address Offset 0B8H PCI Configuration Offset B8 - BBH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																
Bit	Default	Description														
31:13	0	MSI-X Table Offset: Indicates the starting address of the MSI-X Table relative to the address in the Base Address Register indicated bits [2:0] of this register. This part of the MSI-X Table Offset field is programmable and is based on the value programmed in the "MU Base Address Register - MUBAR" on page 444 and the ATU Limit Value Register. The following equation may be used to compute MSI-X Table Offset[31:13]. Note that the Messaging Unit occupies 8-KByte of address space and must overlap the address space defined by the ATU Value and the ATU Limit registers. Equation: MSI-X Table Offset[31:13] = {(~ATU Limit_Register[31:0] & MU_Bar[31:0]) >> 13}. Note: The default location of the MU space after reset is in the first 8-KByte of the default ATU Translation Window.														
12:3	1_1000_0000_0 ₂	MSI-X Table Offset: Indicates the starting address of the MSI-X Table relative to the address in the Base Address Register indicated bits [2:0] of this register. This part of the MSI-X Table Offset field is fixed which forces the table to offset at a 6-KByte offset relative to the "MU Base Address Register - MUBAR" on page 444.														
2:0	000 ₂	PBA BAR Indication Register (BIR): indicates which Base Address Register of the ATU function the Pending Bit Array is mapped into. <table><tr><th>BIR Value</th><th>Base Address Register</th></tr><tr><td>0</td><td>10H</td></tr><tr><td>1</td><td>14H</td></tr><tr><td>2</td><td>18H</td></tr><tr><td>3</td><td>1CH</td></tr><tr><td>4</td><td>20H</td></tr><tr><td>5</td><td>24H</td></tr></table> All other values are reserved.	BIR Value	Base Address Register	0	10H	1	14H	2	18H	3	1CH	4	20H	5	24H
BIR Value	Base Address Register															
0	10H															
1	14H															
2	18H															
3	1CH															
4	20H															
5	24H															

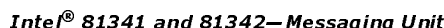


4.9.41 MU MSI-X Control Register X — MMCRx

By default, the MU can generate up to eight MSI-X messages. The MMCRx register provides a control bit that allows collapsing the eight MSI-X messages down to only a single message. When the Host processor cannot honor the eight requested MSI-X messages, the MU MSI-X Single Message Vector bit can be set to cause only a single MSI-X message to be generated.

Table 305. MU MSI-X Control Register X — MMCRx

Register Offset +0BCH Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:01	0000 0000H	Reserved
00	0 ₂	MU MSI-X Single Message Vector: This bit when set, causes only a single MSI-X Message to be generated when MSI-X is enabled. This bit affects the default value of the MSI-X Table Size field in the MSI-X Message Control Register - MSI-X_MCR .



The Inbound MSI Interrupt Pending register is a 32-bit register that is used to post the one-hot decoded bits that are generated by the MU (Messaging Unit) when receiving inbound MSI (Message-Signaled Interrupt). The MU can generate up to 128 interrupts. Refer to the MSI Inbound Message Register — MIMR in the MU Chapter. Any bit set in this register generates an interrupt to the Intel XScale® processor via the Inbound MSI Interrupt pending signal. Software must clear this register to clear any pending interrupt.

Intel XScale® processor Local Bus Address Offset coreID0 - n/a coreID1 - n/a Intel XScale® processor Coprocessor Address IMIPR 0: CP6, CRm 1, CRn 8 IMIPR 1: CP6, CRm 1, CRn 9 IMIPR 2: CP6, CRm 1, CRn 10 IMIPR 3: CP6, CRm 1, CRn 11 Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	0000 0000H	Inbound MSI Pending Interrupts — Each bit reflects an inbound MSI interrupt that was generated by the MU (Messaging Unit). The MU receives, decodes and generates a one-hot bit for every MSI message that it receives. Any bit set generates an interrupt to the Intel XScale® processor via the Inbound MSI Interrupt Pending signal.

The software is responsible for initializing the Circular Queue Size in the MU Configuration Register and all head and tail pointer registers before setting the Circular Queue Enable bit.



5.0 Application DMA Unit

This chapter describes the integrated Application DMA (ADMA) Unit. Operation modes, setup, external interface, and ADMA unit implementation are detailed in this chapter.

5.1 Overview

Application DMA provides low-latency, high-throughput data transfer capability between the ADMA unit, Intel® 81341 and 81342 I/O Processors (81341 and 81342) local memory and host I/O interface (PCI or PCI Express). It executes data transfers from-and-to the 81341 and 81342 local memory, from the host I/O interface to the 81341 and 81342 local memory, or from the 81341 and 81342 local memory to the host I/O interface. ADMA can perform a byte-wise XOR operation across up to 16 concurrent data streams and/or a Galois field multiply of up to 16 concurrent data streams, followed by an XOR operation across them. In addition, the ADMA unit computes a zero result buffer check across local or Host memory blocks, provides a hardware assist to the iSCSI* application by calculating CRC-32C on a single data stream, and performs memory block fills. ADMA is fully programmable from 81341 and 81342 and features:

- Three independent channels
- 6K-byte data transfer queue.
- 2^{36} addressing range on the Internal Bus interface
- 2^{36} addressing range on the DDR SDRAM MCU Port
- 2^{64} addressing range on the Host interface
- Hardware support for unaligned data transfers for both the host I/O interface and the Internal Bus

The Application DMA performs the following functions:

- Three Options for Transferring Data:
 - From memory controller to memory controller
 - From host I/O interface to memory controller
 - From memory controller to host I/O interface
- While Transferring Data the ADMA unit can:
 - Perform an XOR operation on up to 16 read data sources.
 - Perform an optimized dual-XOR calculation for dual XOR-based RAID 6 applications
 - Calculate CRC-32C on the transferred data stream
 - Apply a Galois Field (GF) Multiply function on the Source Data Streams to simultaneously generate P and Q in support of a P+Q RAID6 algorithm.
- The ADMA Unit can verify the integrity of data by:
 - Computing zero result buffer check across memory blocks.
- Perform memory block fills.



5.2 Theory of Operation

The Application DMA provides three channels that perform data transfers from and to the 81341 and 81342 local memory, from the host I/O interface to the 81341 and 81342 local memory, or from the 81341 and 81342 local memory to the host I/O interface. Each channel uses direct addressing for both the memory controller and the host I/O interface.

Each channel can perform memory-to-memory, host-to-memory, or memory-to-host transfers of data blocks.

Each channel implements an XOR algorithm in hardware. It performs an XOR operation on multiple blocks of source (incoming) data and stores results back in either 81341 and 81342 local memory or to the host I/O interface. The source and destination addresses are specified through chain descriptors resident in either 81341 and 81342 DDR SDRAM memory or on the host I/O interface.

Each channel implements a dual-XOR operation that has been optimized for RAID 6 applications. For a single strip RAID 6 write, the dual-XOR operation generates the Horizontal and Diagonal Parity results using new data, old data, old horizontal parity, and old diagonal parity.

A GF Multiply calculation can be applied to source data in support of P+Q RAID-6. The ADMA performs a GF Multiply between source data and a control byte for each source before the XOR operation to generate Q and perform an XOR operation to generate P. P+Q RAID-6 is enabled through the “[P+Q Chain Descriptor Format](#)” by setting an enable bit in the “[ADMA Descriptor Control Register x — ADCRx](#)”.

Each channel checks for a zero result buffer across memory blocks or fill a memory block with arbitrary data.

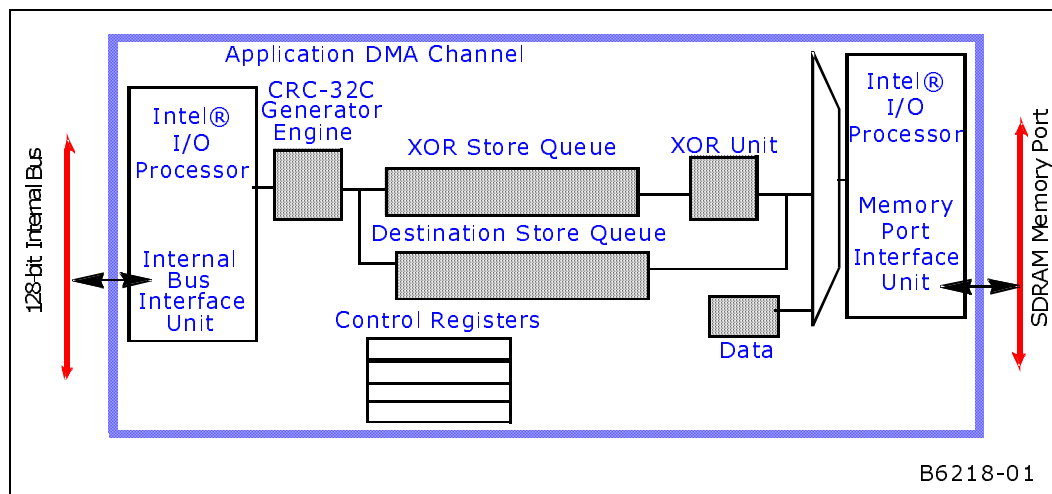
Each channel contains a hardware data alignment unit. This unit enables data transfers from or to unaligned addresses in either the Host address space or the I/O processor local address space. All combinations of unaligned data are supported with the data alignment unit.

Each channel includes a CRC Engine to calculate the CRC-32C algorithm on the data stream being transferred. The CRC engine also initiates the read of a CRC seed and the subsequent write back of the 32-bit result to a CRC Address specified in the descriptor. In addition, a Transfer Complete status bit is updated to the descriptor in memory.



Figure 47 shows a block diagram of the ADMA unit.

Figure 47. Application DMA Channel Block Diagram



Application DMA programming interface is accessible from the internal bus through a memory-mapped register interface. Data for the XOR operation is configured by writing the source addresses, destination address, number of bytes to transfer, and various control information into a chain descriptor. Chain descriptors are described in detail in [Section 5.3, "ADMA Descriptors" on page 466](#).



5.3 ADMA Descriptors

The ADMA operations are managed through the use of descriptors that are arranged in local memory. These descriptors are fetched by the ADMA through pointers contained in the descriptor (in other words, scatter/gather chain descriptors).

The Application DMA Unit implements the XOR algorithm and an XOR algorithm combined with a Galois field multiply operation. It performs these algorithms on multiple streams of source (incoming) data and stores the result back in 81341 and 81342 local memory or to the host I/O interface.

- The process of reading source data, executing the XOR algorithm, and storing the XOR data is hereafter referred to as *XOR-transfer*.
- The process of performing two *XOR-transfer* operations in a single descriptor is hereafter referred to as *Dual-XOR-transfer*. The process of reading or writing data is hereafter referred to as *data transfer*.
- The process of reading source data, performing a Galois field multiply on the source data, then, executing the XOR algorithm, and storing the XOR data is hereafter referred to as *P+Q-transfer*.

The source and destination addresses are specified through the chain descriptors resident in 81341 and 81342 local memory.

There are four types of chain descriptors, Basic, Full, Dual XOR, and P+Q Update. Basic chain descriptors consume 32 bytes of memory, Full chain descriptors consume 152 bytes of memory, while Dual XOR and P+Q Update chain descriptors consume 64 bytes of memory. Table 307 illustrates the different functionality supported by the four descriptor types.

Table 307. Basic and Full Chain Descriptor Functions Support

Type	Data Transfer	CRC-32C Acceleration ^a	Memory Block Fill ^b	XOR Transfer (16 Sources)
Basic	Yes	Yes	Yes	No
Full	No ^c	No	No	Yes
P+Q	No ^d	No	No	Yes
Dual XOR	Supports Dual XOR Operation only			
P+Q Update	Supports P+Q Update Transfer Operation only			

a. CRC-32C acceleration can be performed with/without a Data Transfer.

b. Memory Block Fill can not be combined with any other operation.

c. When the Source Selection field of the ADCR is programmed for more than a single source requiring a Full Descriptor, then the ADMA performs either an XOR Transfer or a Zero Result Buffer Check operation.

d. When the Source Selection field of the ADCR is programmed for more than a single source and the P+Q functionality is enabled, then the ADMA performs either an XOR Transfer with the P+Q function applied to the Source Data streams or a Zero Result Buffer Check operation.



5.3.1 Basic Chain Descriptor Format

The Basic descriptor can be used to perform a *data transfer* with the option of performing the CRC-32C calculation on the transferred data stream; also, the CRC-32C calculation may be performed without a *data transfer*. In addition, the Basic Descriptor supports the Memory Block Fill operation.

To perform a transfer, one or more chain descriptors must first be written to memory. Figure 48 shows the format of an individual basic chain descriptor. Every basic descriptor requires eight contiguous words in local memory and is required to be aligned on a 32-byte address boundary. All eight words are required.

Figure 48. Basic Chain Descriptor Format

Next Descriptor Address	Word 0
Descriptor Control	Word 1
CRC Address (Lower) / Memory Block Fill Data	Word 2
Byte Count	Word 3
Destination Address (Lower)	Word 4
Destination Address (Upper)	Word 5
Source 0 Address (Lower)	Word 6
Source 0 Address (Upper)	Word 7

B6219-01



Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- The first word is the local memory address of the next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the ADMA Next Descriptor Address Register. Because basic chain descriptors must be aligned on an 8-word boundary, the unit may ignore bits 04:00 of this address.
- The second word is the Descriptor Control Word. This word configures the Application DMA for one operation. This value is loaded into the ADMA Descriptor Control Register.
- The third word is the CRC Address / Memory Block Fill Data word. When the CRC operation is enabled, the CRC address is used to load a 32-bit value to seed the CRC calculation. Also, the CRC address is used as the destination for the write back of the CRC result. When the Memory Block fill operation is enabled, this Register is used as the immediate data to fill memory with. This value is loaded into the CRC Address / Memory Block Fill Data Register.
- The fourth word contains the 24-bit, Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. Also, the upper byte of this word is the Transfer Status field that may be written back to the Byte Count Word at the end of the ADMA transfer when the Status Write Back Enable in the ADCR is set. This value is loaded into the ADMA Byte Count Register.
- The fifth word is the lower 32-bit destination address. The ADMA uses this value for the lower 32-bits of the destination address for the *data transfer* operation, the Memory Block Fill operation, and the *XOR-transfer* operation. This value is loaded into the Destination Lower Address Register.
- The sixth word is the upper 32-bit destination address. The ADMA uses this value for the upper 32-bits of the destination address for the *data transfer* operation, the Memory Block Fill operation, and the *XOR-transfer* operation. This value is loaded into the Destination Upper Address Register.
- The seventh word is the lower 32-bit source address for the *data transfer* operation or the lower 32-bit source address for source 0 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 0.
- The eighth word is the upper 32-bit source address for the *data transfer* operation or the upper 32-bit source address for source 0 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 0.



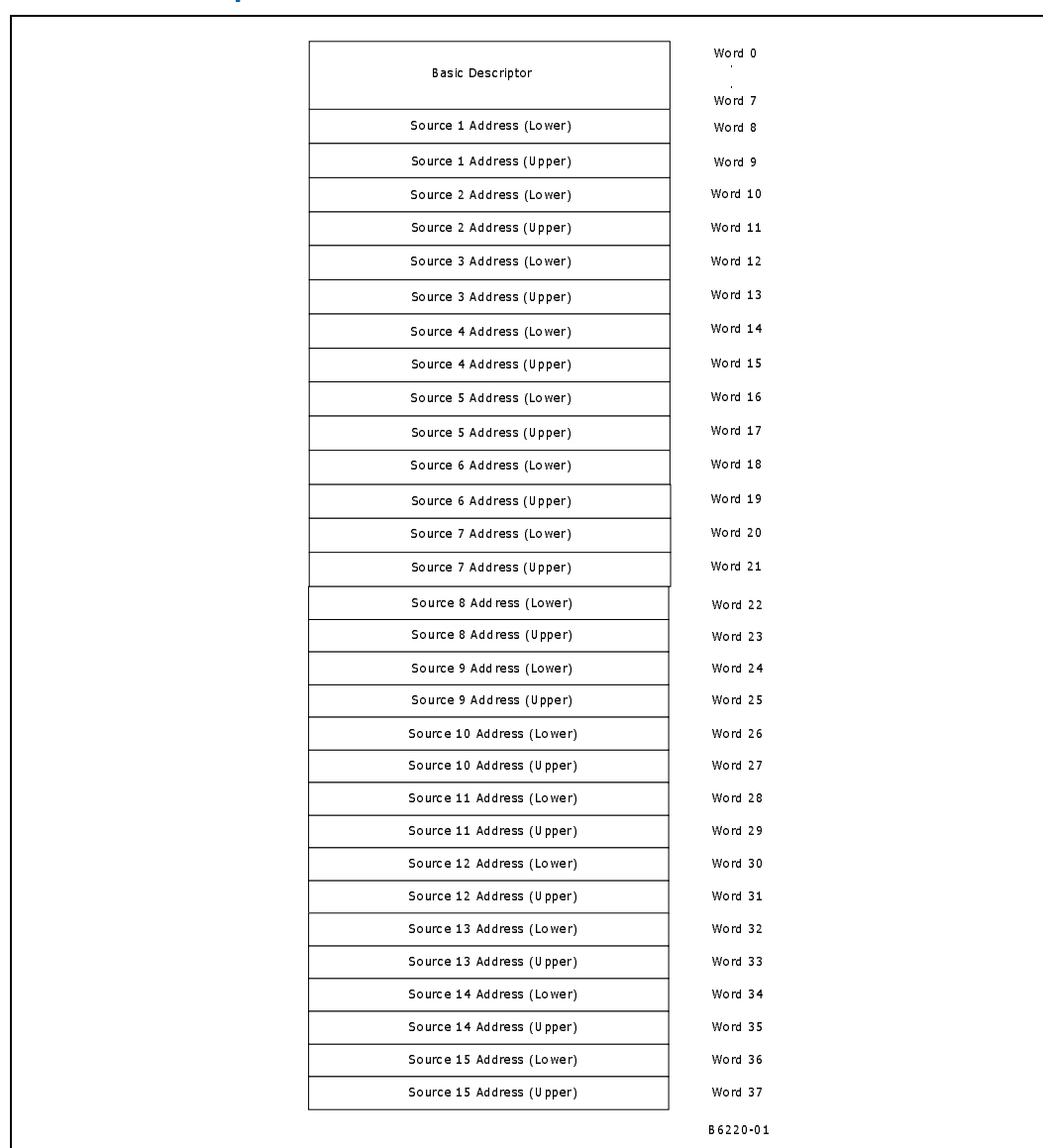
5.3.2 Full Chain Descriptor Format

The Full descriptor can be used to perform an *XOR-transfer* for up to 16 sources.

To perform a transfer, one or more chain descriptors must first be written to memory. [Figure 49](#) and [Figure 50 on page 473](#) show the format of an individual full chain descriptor. Note that the full descriptor is a superset of the Basic descriptor. The functions programmed into the Descriptor Control word (DC) for the current descriptor determines whether the 30 extra words required by the full descriptor are fetched by the ADMA (see [Table 307, “Basic and Full Chain Descriptor Functions Support” on page 466](#)).

Every Full descriptor may contain up to 38 contiguous words in the local memory. The Full Descriptor is required to be aligned on a 32 byte address boundary.

Figure 49. Full Chain Descriptor Format





Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- The first eight words are the Basic descriptor. Please see [Section 5.3.1, “Basic Chain Descriptor Format”](#) on page 467 for more details.
- The ninth word is the lower 32-bit source address for source 1 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 1.
- The tenth word is the upper 32-bit source address for source 1 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 1.
- The eleventh word is the lower 32-bit source address for source 2 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 2.
- The twelfth word is the upper 32-bit source address for source 2 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 2.
- The thirteenth word is the lower 32-bit source address for source 3 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 3.
- The fourteenth word is the upper 32-bit source address for source 3 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 3.
- The fifteenth word is the lower 32-bit source address for source 4 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 4.
- The sixteenth word is the upper 32-bit source address for source 4 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 4.
- The seventeenth word is the lower 32-bit source address for source 5 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 5.
- The eighteenth word is the upper 32-bit source address for source 5 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 5.
- The nineteenth word is the lower 32-bit source address for source 6 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 6.
- The twentieth word is the upper 32-bit source address for source 6 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 6.
- The twenty first word is the lower 32-bit source address for source 7 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 7.
- The twenty second word is the upper 32-bit source address for source 7 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 7.
- The twenty third word is the lower 32-bit source address for source 8 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 8.
- The twenty fourth word is the upper 32-bit source address for source 8 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 8.
- The twenty fifth word is the lower 32-bit source address for source 9 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 9.
- The twenty sixth word is the upper 32-bit source address for source 9 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 9.
- The twenty seventh word is the lower 32-bit source address for source 10 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 10.
- The twenty eighth word is the upper 32-bit source address for source 10 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 10.
- The twenty ninth word is the lower 32-bit source address for source 11 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 11.



- The thirtieth word is the upper 32-bit source address for source 11 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 11.
- The thirty first word is the lower 32-bit source address for source 12 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 12.
- The thirty second word is the upper 32-bit source address for source 12 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 12.
- The thirty third word is the lower 32-bit source address for source 13 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 13.
- The thirty fourth word is the upper 32-bit source address for source 13 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 13.
- The thirty fifth word is the lower 32-bit source address for source 14 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 14.
- The thirty sixth word is the upper 32-bit source address for source 14 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 14.
- The thirty seventh word is the lower 32-bit source address for source 15 of the *XOR-transfer* operation. This value is loaded into the Source Lower Address Register 15.
- The thirty eighth word is the upper 32-bit source address for source 15 of the *XOR-transfer* operation. This value is loaded into the Source Upper Address Register 15.

5.3.3 P+Q Chain Descriptor Format

The P+Q chain descriptor can be used to perform an *P+Q-transfer* while performing a Galois field multiply function on the all of the source data streams; this operation is referred to as a *P+Q-transfer*. See [Section 5.7.3, "XOR Operation with P+Q RAID-6"](#) on [page 494](#) for more details.

To perform a transfer, one or more chain descriptors must first be written to memory. [Figure 50](#) shows the format of an individual P+Q chain descriptor. Every P+Q chain descriptor requires a minimum of eight contiguous words in the local memory and is required to be aligned on a 32-byte address boundary. While a minimum of eight words are required, a maximum of 38 words may be used to perform the P+Q function on up to 16 source data streams.

Note: The minimum size (number of source = 1) P+Q chain descriptor is only useful for performing a Galois field multiply on a single source data stream and writing it to the Q Destination Address (e.g., 'P Transfer Disable' bit in the ADCRx is set).

Warning: The hardware requires that bits 3 down to 0 of the Q_Destination Address (Word 2 and 5) and the P_Destination Address (Word 4 and 5) are programmed to the same four-bit value. Furthermore, the hardware requires this even if the **P Transfer Disable** bit is set to disable P transfer to the P_Destination Address. For example, in this case the P_Destination Address may contain an invalid address such that bits 3 down to 0 of that address equals to bits 3 down to 0 of the Q_Destination Address.

**Figure 50. P+Q Chain Descriptor Format (Minimum Size)**

Next Descriptor Address		Word0
Descriptor Control		Word1
Q_Destination Address (Lower)		Word2
Byte Count		Word3
P_Destination Address (Lower)		Word4
PQ_Destination Address (Upper)		Word5
Source 0 Address (Lower)		Word6
DMLT0	Source 0 Address (Upper)	Word7
Source 1 Address (Lower)		Word 8
DMLT1	Source 1 Address (Upper)	Word 9
Source 2 Address (Lower)		Word 10
DMLT2	Source 2 Address (Upper)	Word 11
Source 3 Address (Lower)		Word 12
DMLT3	Source 3 Address (Upper)	Word 13
Source 4 Address (Lower)		Word 14
DMLT4	Source 4 Address (Upper)	Word 15
Source 5 Address (Lower)		Word 16
DMLT5	Source 5 Address (Upper)	Word 17
Source 6 Address (Lower)		Word 18
DMLT6	Source 6 Address (Upper)	Word 19
Source 7 Address (Lower)		Word 20
DMLT7	Source 7 Address (Upper)	Word 21
Source 8 Address (Lower)		Word 22
DMLT8	Source 8 Address (Upper)	Word 23
Source 9 Address (Lower)		Word 24
DMLT9	Source 9 Address (Upper)	Word 25
Source 10 Address (Lower)		Word 26
DMLT10	Source 10 Address (Upper)	Word 27
Source 11 Address (Lower)		Word 28
DMLT11	Source 11 Address (Upper)	Word 29
Source 12 Address (Lower)		Word 30
DMLT12	Source 12 Address (Upper)	Word 31
Source 13 Address (Lower)		Word 32
DMLT13	Source 13 Address (Upper)	Word 33
Source 14 Address (Lower)		Word 34
DMLT14	Source 14 Address (Upper)	Word 35
Source 15 Address (Lower)		Word 36
DMLT15	Source 15 Address (Upper)	Word 37

B6221-01

Each word in P+Q chain descriptor is analogous to control register values. Bit definitions for words in the chain descriptor are the same as for the control registers.

- First word is local memory address of next chain descriptor. A value of '0' specifies the end of chain. This value is loaded into the ADMA Next Descriptor Address Register. Because basic chain descriptors must be aligned on an 8-word boundary, the unit may ignore bits 04:00 of this address.



- Second word is Descriptor Control Word. This word configures the Application DMA for one operation. This value is loaded into the ADMA Descriptor Control Register.
- Third word is lower 32-bit Q destination address for the *P+Q-transfer* operation. This value is loaded into the CRC Address / Memory Block Fill Data Register.
- Fourth word contains 24-bit, Byte Count value, which specifies the number of data bytes in the current chain descriptor. The upper byte of this word is the Transfer Status field that may be written back to the Byte Count Word at the end of the ADMA transfer, when the ADCR Status Write Back Enable is set. This value is loaded into the ADMA Byte Count Register.
- Fifth word is lower 32-bit destination address. The ADMA uses this value for the lower 32-bits of the destination address for the *P+Q-transfer* operation, the Memory Block Fill operation, and the *P+Q-transfer* operation. This value is loaded into the Destination Lower Address Register.
- Sixth word is upper 32-bit destination address. The ADMA uses this value for the upper 32-bits of the destination address for the *P+Q-transfer* operation, the Memory Block Fill operation, and the *P+Q-transfer* operation. This value is loaded into the Destination Upper Address Register.
- Seventh word is lower 32-bit source address for the *P+Q-transfer* operation or the lower 32-bit source address for source 0 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 0.
- Eighth word is upper 4-bit source address and 8-bit Galois Field Multiplier for source 0 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 0.
- Ninth word is lower 32-bit source address for source 1 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 1.
- Tenth word is upper 4-bit source address and 8-bit Galois Field Multiplier for the *P+Q-transfer* source 1 operation. This value is loaded into the Source Upper Address Register 1.
- Eleventh word is lower 32-bit source address for source 2 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 2.
- Twelfth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 2 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 2.
- Thirteenth word is lower 32-bit source address for source 3 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 3.
- Fourteenth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 3 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 3.
- Fifteenth word is lower 32-bit source address for source 4 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 4.
- Sixteenth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 4 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 4.
- Seventeenth word is lower 32-bit source address for source 5 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 5.
- Eighteenth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 5 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 5.
- Nineteenth word is lower 32-bit source address for source 6 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 6.



- Twentieth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 6 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 6.
- Twenty first word is lower 32-bit source address for source 7 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 7.
- Twenty second word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 7 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 7.
- Twenty third word is lower 32-bit source address for source 8 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 8.
- Twenty fourth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 8 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 8.
- Twenty fifth word is lower 32-bit source address for source 9 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 9.
- Twenty sixth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 9 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 9.
- Twenty seventh word is lower 32-bit source address for source 10 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 10.
- Twenty eighth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 10 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 10.
- Twenty ninth word is lower 32-bit source address for source 11 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 11.
- Thirtieth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 11 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 11.
- Thirty first word is lower 32-bit source address for source 12 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 12.
- Thirty second word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 12 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 12.
- Thirty third word is lower 32-bit source address for source 13 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 13.
- Thirty fourth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 13 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 13.
- Thirty fifth word is lower 32-bit source address for source 14 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 14.
- Thirty sixth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 14 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 14.
- Thirty seventh word is lower 32-bit source address for source 15 of the *P+Q-transfer* operation. This value is loaded into the Source Lower Address Register 15.
- Thirty eighth word is upper 4-bit source address and the 8-bit Galois Field Multiplier for source 15 of the *P+Q-transfer* operation. This value is loaded into the Source Upper Address Register 15.



5.3.4 Dual XOR Descriptor Format

The Dual XOR descriptor can be used to perform a *Dual-XOR-transfer*. To perform a *Dual-XOR-transfer*, the Dual XOR format descriptor must be written to local memory. [Figure 51](#) shows the format of an individual Dual XOR chain descriptor. Every Dual XOR descriptor requires 16 contiguous words in local memory and is required to be aligned on a 32-byte address boundary. All 16 words are required.

Note: Bits 3 down to 0 of the Horizontal Destination Address (Word 4 and 5) and the Diagonal Destination Address (Word 14 and 15) must be programmed to the same four-bit value.

Figure 51. Dual XOR Descriptor Format

Next Descriptor Address	Word 0
Descriptor Control	Word 1
Reserved (00000000h)	Word 2
Byte Count	Word 3
H - Destination Address (Lower)	Word 4
H - Destination Address (Upper)	Word 5
Source 0 Address (Lower)	Word 6
Source 0 Address (Upper)	Word 7
Source 1 Address (Lower)	Word 8
Source 1 Address (Upper)	Word 9
H - Source Address (Lower)	Word 10
H - Source Address (Upper)	Word 11
D - Source Address (Lower)	Word 12
D - Source Address (Upper)	Word 13
D - Destination Address (Lower)	Word 14
D - Destination Address (Upper)	Word 15

B6222-01



Each word in the Dual XOR chain descriptor is analogous to control register values. A chain descriptor is interpreted as a Dual XOR operation when bit 17 of the second word in the descriptor is set; note that this is analogous to bit 17 of the “ADMA Descriptor Control Register x — ADCRx” on page 526.

- First word is the local memory address of the next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the ADMA Next Descriptor Address Register. Because basic chain descriptors must be aligned on an 8-word boundary, the unit may ignore bits 04:00 of this address.
- Second word is the Descriptor Control Word. This word configures Application DMA for one operation. This value is loaded into the ADMA Descriptor Control Register.
- Third word is Reserved and must be written with 0000 0000H.
- Fourth word contains the 24-bit, Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. Also, the upper byte of this word is the Transfer Status field that may be written back to the Byte Count Word at the end of the ADMA transfer when the Status Write Back Enable in the ADCR is set. This value is loaded into the ADMA Byte Count Register.
- Fifth word is the lower 32-bit Horizontal destination address. The ADMA uses this value for the lower 32-bits of the Horizontal destination address for the *Dual-XOR-transfer* operation. This value is loaded into the Destination Lower Address Register.
- Sixth word is the upper 32-bit Horizontal destination address. The ADMA uses this value for the upper 32-bits of the Horizontal destination address for the *Dual-XOR-transfer* operation. This value is loaded into the Destination Upper Address Register.
- Seventh word is lower 32-bit source address for first data block of the *Dual-XOR-transfer* operation. This value is loaded into the Source Lower Address Register 0.
- Eighth word is upper 32-bit source address for the first data block of the *Dual-XOR-transfer* operation. This value is loaded into the Source Upper Address Register 0.
- Ninth word is lower 32-bit source address for the second data block of the *Dual-XOR-transfer* operation. This value is loaded into the Source Lower Address Register 1.
- Tenth word is upper 32-bit source address for the second data block of the *Dual-XOR-transfer* operation. This value is loaded into the Source Upper Address Register 1.
- Eleventh word is lower 32-bit source address for Horizontal source of the *Dual-XOR-transfer* operation. This value is loaded into the Source Lower Address Register 2.
- Twelfth word is upper 32-bit source address for the Horizontal source of the *Dual-XOR-transfer* operation. This value is loaded into the Source Upper Address Register 2.
- Thirteenth word is lower 32-bit source address for Diagonal source of the *Dual-XOR-transfer* operation. This value is loaded into the Source Lower Address Register 3.
- Fourteenth word is upper 32-bit source address for the Diagonal source of the *Dual-XOR-transfer* operation. This value is loaded into the Source Upper Address Register 3.
- Fifteenth word is lower 32-bit destination address for the Diagonal destination of the *Dual-XOR-transfer* operation. This value is loaded into the Source Lower Address Register 4.
- Sixteenth word is upper 32-bit destination address for the Diagonal destination of the *Dual-XOR-transfer* operation. This value is loaded into the Source Upper Address Register 4.



5.3.5 P+Q Update Descriptor Format

The P+Q Update descriptor can be used to perform a *P+Q-Update-Transfer*. To perform a *P+Q-Update-Transfer*, the P+Q Update format descriptor must be written to local memory. [Figure 51 on page 476](#) shows the format of an individual P+Q Update chain descriptor. Every P+Q Update descriptor requires 16 contiguous words in local memory and is required to be aligned on a 32-byte address boundary. All 16 words are required.

Warning: The hardware requires that bits 3 down to 0 of the Q_Destination Address (Word 2 and 5) and the P_Destination Address (Word 4 and 5) are programmed to the same four-bit value. Furthermore, the hardware requires this even if the **P Transfer Disable** bit is set to disable P transfer to the P_Destination Address. For example, in this case the P_Destination Address may contain an invalid address such that bits 3 down to 0 of that address equals to bits 3 down to 0 of the Q_Destination Address.

Figure 52. P+Q Update Descriptor Format

Next Descriptor Address		Word 0
Descriptor Control		Word 1
Reserved (00000000h)		Word 2
Byte Count		Word 3
P - Destination Address (Lower)		Word 4
P - Destination Address (Upper)		Word 5
Source 0 Address (Lower)		Word 6
Source 0 Address (Upper)		Word 7
Source 1 Address (Lower)		Word 8
Source 1 Address (Upper)		Word 9
P - Source Address (Lower)		Word 10
P - Source Address (Upper)		Word 11
Q - Source Address (Lower)		Word 12
DMLTQ	Q - Source Address (Upper)	Word 13
Q - Destination Address (Lower)		Word 14
Q - Destination Address (Upper)		Word 15

B6223-01



Each word in the P+Q Update chain descriptor is analogous to control register values. A chain descriptor is interpreted as a P+Q Update operation when bit 16 of the second word in the descriptor is set; note that this is analogous to bit 16 of the “ADMA Descriptor Control Register x — ADCRx” on page 526.

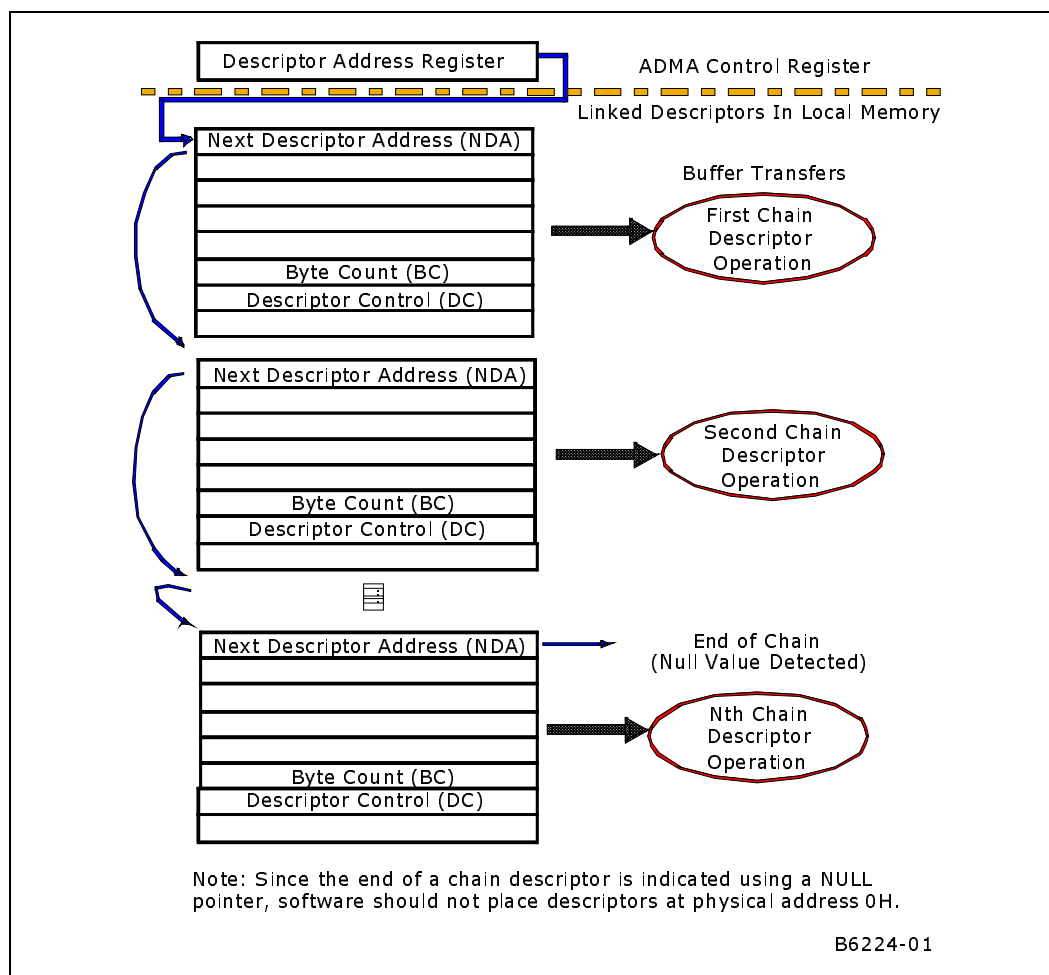
- First word is the local memory address of the next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the ADMA Next Descriptor Address Register. Because basic chain descriptors must be aligned on an 8-word boundary, the unit may ignore bits 04:00 of this address.
- Second word is the Descriptor Control Word. This word configures the Application DMA for one operation. This value is loaded into the ADMA Descriptor Control Register.
- Third word is Reserved and must be written with 0000 0000H.
- Fourth word contains the 24-bit, Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. Also, the upper byte of this word is the Transfer Status field that may be written back to the Byte Count Word at the end of the ADMA transfer when the Status Write Back Enable in the ADCR is set. This value is loaded into the ADMA Byte Count Register.
- Fifth word is the lower 32-bit P destination address. The ADMA uses this value for the lower 32-bits of the P destination address for the *P+Q-Update-Transfer* operation. This value is loaded into the Destination Lower Address Register.
- Sixth word is the upper 32-bit P destination address. The ADMA uses this value for the upper 32-bits of the P destination address for the *P+Q-Update-Transfer* operation. This value is loaded into the Destination Upper Address Register.
- Seventh word is the *P+Q-Update-Transfer* operation lower 32-bit source address for the first data block. This value is loaded into the Source Lower Address Register 0.
- Eighth word is the *P+Q-Update-Transfer* operation upper 32-bit source address for the first data block. This value is loaded into the Source Upper Address Register 0.
- Ninth word is the *P+Q-Update-Transfer* operation lower 32-bit source address for the second data block. This value is loaded into the Source Lower Address Register 1.
- Tenth word is the *P+Q-Update-Transfer* operation upper 32-bit source address for the second data block. This value is loaded into the Source Upper Address Register 1.
- Eleventh word is the lower 32-bit source address for P source of the *P+Q-Update-Transfer* operation. This value is loaded into the Source Lower Address Register 2.
- Twelfth word is the *P+Q-Update-Transfer* operation upper 32-bit source address for the P source. This value is loaded into the Source Upper Address Register 2.
- Thirteenth word is the *P+Q-Update-Transfer* operation lower 32-bit source address for Q source. This value is loaded into the Source Lower Address Register 3.
- Fourteenth word is the upper 32-bit source address and the 8-bit Q Update Multiplier for the Q source of the *P+Q-Update-Transfer* operation. This value is loaded into the Source Upper Address Register 3.
- Fifteenth word is the *P+Q-Update-Transfer* operation lower 32-bit destination address for the Q destination. This value is loaded into the Source Lower Address Register 4.
- Sixteenth word is the *P+Q-Update-Transfer* operation upper 32-bit destination address for the Q destination. This value is loaded into the Source Upper Address Register 4.

5.4 ADMA Descriptor Processing

To perform an *XOR-transfer*, a series of chain descriptors can be built in local memory to XOR multiple blocks of source data resident in local memory. The XOR-ed result is then stored back to either Host or local memory. An application can build multiple chain descriptors to XOR many blocks of data which have different source addresses within the local or Host memory.

When multiple chain descriptors are built in local or SRAM memory, the application can link each of these chain descriptors using the Next Descriptor Address in the chain descriptor. This address logically links the chain descriptors together. This allows the application to build a list of transfers which may not require the processor until all transfers are complete. Figure 53 shows an example of a linked-list of transfers specified in external memory.

Figure 53. Chaining Operation





The following describes the steps for initiating a new ADMA operation:

1. ADMA must be inactive prior to starting an ADMA operation. This can be checked by software by reading the *ADMA Active* bit in the ADMA Channel Status Register. When this bit is clear, the unit is inactive. When this bit is set, the unit is currently active.
2. The ACSR must be cleared of all error conditions.
3. The software writes the address of the first chain descriptor to the ADMA Next Descriptor Address Register (ANDAR).
4. The software sets the *ADMA Enable* bit in the ADMA Channel Control Register (ACCR). Because this is the start of a new ADMA operation and not the resumption of a previous operation, the *Chain Resume* bit in the ACCR should be clear.
5. The ADMA starts the ADMA operation by reading the chain descriptor at the address contained in ANDAR. The ADMA loads the chain descriptor values into the ADAR and begins data transfer. The ADMA Descriptor Address Register (ADAR) contains the address of the chain descriptor just read and ANDAR now contains the Next Descriptor Address from the chain descriptor just read.

The last descriptor in the ADMA chain list has zero in the next descriptor address field specifying the last chain descriptor. A NULL value notifies the ADMA not to read additional chain descriptors from memory.

Once an ADMA operation is active, it can be temporarily suspended by clearing the *ADMA Enable* bit in the ACCR. Note that this does not abort the ADMA operation. The unit resumes the process when the *ADMA Enable* bit is set.

When descriptors are read from external memory, bus latency and memory speed affect chaining latency. Chaining latency is defined as the time required for the ADMA to access the next chain descriptor plus the time required to set up the next ADMA operation.

5.4.1 Scatter Gather Transfers

The Application DMA can be used to perform typical scatter gather transfers. This consists of programming the chain descriptors to gather data which may be located in non-contiguous blocks of memory. The chain descriptor specifies the destination location such that once all data has been processed, the data is contiguous in memory. [Figure 54](#) shows how the destination pointers can gather data.

5.4.2 Synchronizing a Program to Chained Operation

Any operation involving the ADMA can be synchronized to a program executing on the Intel XScale® processor through the use of processor interrupts. The ADMA generates an interrupt to the Intel XScale® processor under certain conditions. They are:

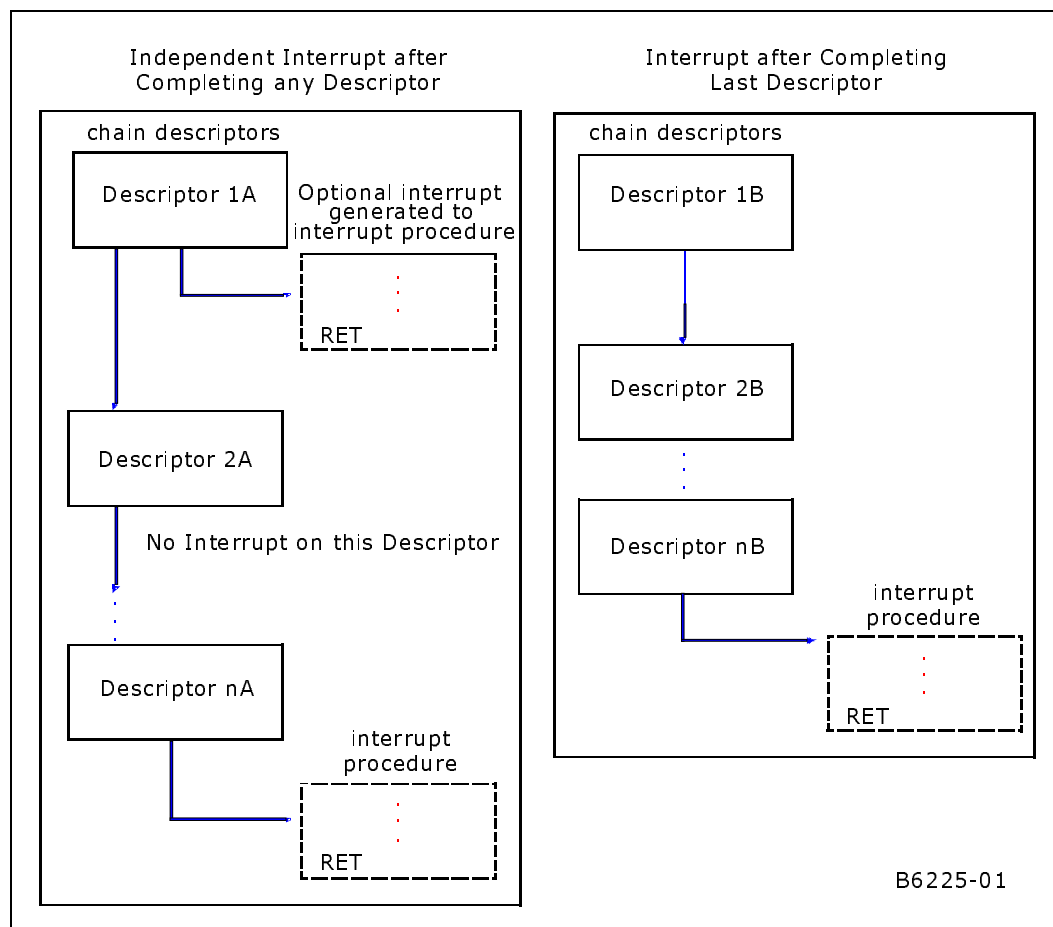
1. [Interrupt & Continue] The ADMA completes processing a chain descriptor and the ADMA Next Descriptor Address Register (ANDAR) is non-zero. When the *Interrupt Enable* bit within the ADMA Descriptor Control Register (ADCR) is set, an interrupt is generated to the Intel XScale® processor. This interrupt is for synchronization purposes. The ADMA sets the *End Of Transfer Interrupt* flag in the ADMA Channel Status Register (ACSR). Since it is not the last chain descriptor in the list, the ADMA starts to process the next chain descriptor without requiring any processor interaction.
2. [End of Chain] The ADMA completes processing a chain descriptor and the ADMA Next Descriptor Address Register is zero specifying the end of the chain. When the *Interrupt Enable* bit within the ADCR is set, an interrupt is generated to the Intel XScale® processor. The ADMA sets the *End Of Chain Interrupt* flag in the ACSR.
3. [Error] An error condition occurs (refer to [Section 5.14, "Error Conditions" on page 519](#) for Application DMA error conditions) during a transfer. The ADMA halts operation on the current chain descriptor and not proceed to the next chain descriptor.

Each chain descriptor can independently set the *Interrupt Enable* bit in the Descriptor Control word. This bit enables an independent interrupt once a chain descriptor is processed. This bit can be set or clear within each chain descriptor. Control of interrupt generation within each descriptor aids in synchronization of the executing software with ADMA operation.



Figure 54 shows two examples of program synchronization. The left column shows program synchronization based on individual chain descriptors. Descriptor 1A generated an interrupt to the processor, while descriptor 2A did not because the *Interrupt Enable* bit was clear. The last descriptor *nA*, generated an interrupt to signify the end of the chain has been reached. The right column in Figure 54 shows an example where the interrupt was generated only on the last descriptor signifying the end of chain.

Figure 54. Synchronizing to Chained ADMA Operation



5.4.3 Appending to the End of a Chain

Once the ADMA has started processing a chain of descriptors, application software may need to append a chain descriptor to the current chain without interrupting the transfer in progress. The mechanism used for performing this action is controlled by the *Chain Resume* bit in the ADMA Control Register (ACCR).

The ADMA reads the subsequent chain descriptor each time it completes the current chain descriptor and the ADMA Next Descriptor Address Register (ANDAR) is non-zero. ANDAR always contains the address of the next chain descriptor to be read and the ADMA Descriptor Address Register (ADAR) always contains the address of the current chain descriptor.

The procedure for appending chains requires the software to find the last chain descriptor in the current chain and change the Next Descriptor Address in that descriptor to the address of the new chain to be appended. The software then sets the *Chain Resume* bit in the ACCR. It does not matter when the unit is active or not.

The ADMA examines the *Chain Resume* bit of the ACCR when the unit is idle or upon completion of a chain of transfers. When this bit is set, the ADMA re-reads the Next Descriptor Address of the current chain descriptor and load it into ANDAR. The address of the current chain descriptor is contained in ADAR. The ADMA clears the *Chain Resume* bit and then examines ANDAR. When ANDAR is not zero, the ADMA reads the chain descriptor using this new address and begin a new operation. When ANDAR is zero, the ADMA remains or return to idle.

There are three cases to consider:

1. The ADMA completes an ADMA operation and it is not the last descriptor in the chain. In this case, the ADMA clears the *Chain Resume* bit and reads the next chain descriptor. The appended descriptor is read when the ADMA reaches the end of the original chain.
2. The channel completes an ADMA transfer and it is the last descriptor in the chain. In this case, the ADMA examines the state of the *Chain Resume* bit. When the bit is set, the ADMA re-reads the current descriptor to get the address of the appended chain descriptor. When the bit is clear, the ADMA returns to idle.
3. The ADMA is idle. In this case, the ADMA examines the state of the *Chain Resume* bit when the ACCR is written. When the bit is set, the ADMA re-reads the last descriptor from the most-recent chain to get the appended chain descriptor.



5.5 Data Queues

ADMA Ch-0 and Ch-1 each contain six 1 Kbyte data buffers. Four of the buffers are used when the Transfer Direction field in the ADCR selects the Local Memory (DDR SDRAM) as the transfer destination ("00", "10"), while the remaining two buffers are used when either the Internal Bus or the Host I/O interface is selected as the transfer destination ("01", "11").

ADMA Ch-2 contains four 1 Kbyte data buffers. Two of the buffers are used when the Transfer Direction field in the ADCR selects the Local Memory (DDR SDRAM) as the transfer destination ("00", "10"), while the remaining two buffers are used when either the Internal Bus or the Host I/O interface is selected as the transfer destination ("01", "11").

These queues temporarily hold data to increase the performance of data transfers in both directions.

5.5.1 Data Transfer Options

The Transfer Direction field in the ADCR is used to specify the source/destination interfaces used by the *data transfer*, the *XOR-transfer*, the *P+Q-transfer*, the Zero Result Buffer Check operation, the P+Q Zero Result Buffer Check Operation, and the Memory Block Fill operation.

The valid options for this field and how they are applied to these operations are specified in [Table 308 on page 486](#). This table details the ADMA address registers associated with the source/destination of a particular operation for a given Transfer Direction field selection.



Note: For Table 308 entries where the descriptor field name does **not** match the register name, the descriptor field name is included in parentheses.

Table 308. ADMA Operation Data Transfer Options

Transfer Direction Field	"00"		"01"		"10"		"11"	
	Source Host I/O Interface	Destination Local Memory	Source Local Memory	Destination Host I/O Interface ^a	Source Internal Bus	Destination Local Memory	Source Local Memory	Destination Internal Bus ^a
Data Transfer ^b	SAR0 ^c	DADR ^d	SAR0	DADR	SAR0	DADR	SAR0	DADR
XOR Transfer	NA	NA	SAR0... SAR15	DADR	NA	NA	SAR0... SAR15	DADR
P+Q Transfer	NA	NA	SAR0... SAR15	DADR (P_DADR) CARMDQ (Q_DADR)	NA	NA	SAR0... SAR15	DADR (P_DADR) CARMDQ (Q_DADR)
Dual XOR Transfer	NA	NA	SAR0... SAR3 ^e	DADR (H_DADR) SAR4 (D_DADR)	NA	NA	SAR0... SAR3	DADR (H_DADR) SAR4 (D_DADR)
P+Q Update Transfer	NA	NA	SAR0... SAR3 ^f	DADR (P_DADR) SAR4 (Q_DADR)	NA	NA	SAR0... SAR3	DADR (P_DADR) SAR4 (Q_DADR)
Zero Result Buffer Check Operation	NA	NA ^g	SAR0... SAR15	NA	NA	NA	SAR0... SAR15	NA
P+Q Zero Result Buffer Check Operation	NA	NA ^h	SAR0... SAR15	NA	NA	NA	SAR0... SAR15	NA
Memory Block Fill Operation	NA	DADR	NA	DADR	NA	DADR	NA	DADR

- a. For table entries where the descriptor field name does not match the register name, the descriptor field name is included in parentheses. For example, the P Destination Address (P_DADR) descriptor field for a P+Q Transfer is mapped to the Destination Address Register (DADR).
- b. Data Transfer with/without CRC Generation. Irrespective of the setting of the Transfer Direction field, the CRC Value is read/written from the Internal Bus using the CRC Address contained in the Basic Descriptor and the upper CRC Address contained in the ACCR. When Data Transfer during a CRC operation is disabled, the data stream is read from the Source interface but not written to the Destination interface.
- c. Source Lower/Upper Address Register 0 pair which is used as the Local Memory Address for the Data Transfer Operation.
- d. Destination Lower/Upper Address Register pair.
- e. For Dual XOR Operation, SAR2, SAR3, and SAR4 represent the Horizontal Source Address, Diagonal Source Address, and Diagonal Destination Address, respectively. SAR0 and SAR1 represent Old Source Data, and the New Source data for the Dual XOR operation, but order is not relevant.
- f. For the P+Q Update Transfer Operation, SAR2, SAR3, and SAR4 represent the P Source Address, Q Source Address (including Q Update Multiplier), and Q Destination Address, respectively. SAR0 and SAR1 represent Old Source Data, and the New Source data for the P+Q Update Transfer operation, but order is not relevant.
- g. Not Applicable
- h. Not Applicable

Note: For Transfer Direction field value "01", the DUADR register specifies the upper 32-bits of a 64-bit host I/O interface address. For Transfer Direction field values "00", "10" and "11", the DUADR register specifies the upper 4-bits of the 36-bit internal bus or local memory address. For Transfer Direction field value "00", the SUAR0...15 registers specify the upper 32-bits of a 64-bit host I/O interface address. For Transfer Direction field values "01", "10" and "11", the SUAR0...15 registers specify the upper 4-bits of the 36-bit internal bus or local memory address.



5.6 Data Alignment

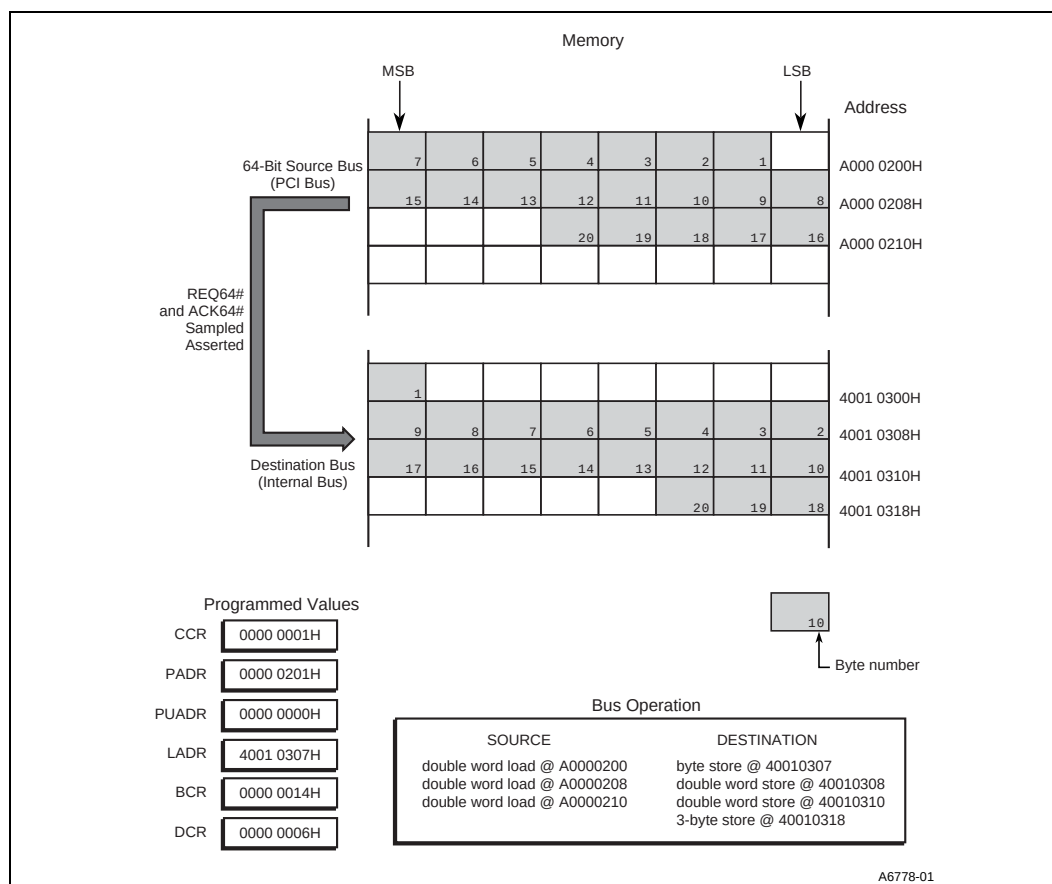
Each channel contains a hardware data alignment unit to support unaligned data transfers between the source and destination busses. The data alignment unit optimizes data transfers to and from 32 and 64-bit memory. The channel reformats data words for the correct bus data width.

Aligned data transfers involve data accesses that fall on natural boundaries. For example; double words are aligned on 8-byte boundaries and words are aligned on 4-byte boundaries. ADMA transfers can occur with both the source and destination addresses unaligned.

5.6.1 64-bit Unaligned Data Transfers

Figure 55 illustrates an ADMA transfer between unaligned 64-bit source and destination addresses.

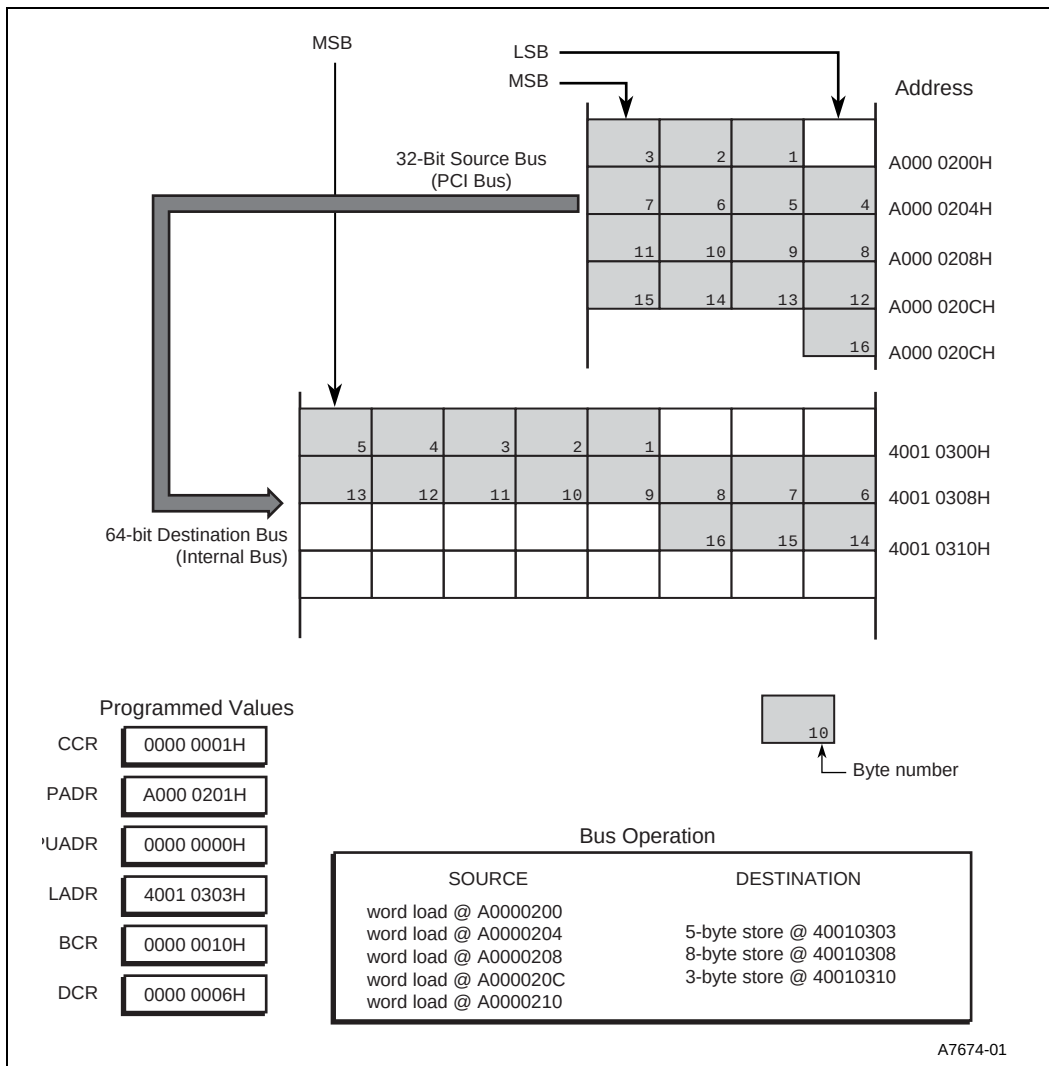
Figure 55. Optimization of an Unaligned ADMA Transfer



64/32-bit Unaligned Data Transfers

Figure 56 illustrates an ADMA transfer between an unaligned 32-bit source address and an unaligned 64-bit destination address.

Figure 56. Optimization of an Unaligned ADMA Transfer





5.7 ADMA Operations

The ADMA can be configured on a per descriptor basis through the Descriptor Control Word (DC) to perform seven distinct operations in addition to a simple data transfer:

1. In an **XOR operation**, the ADMA generates a parity data stream that is comprised of the XOR of up to 16 distinct data streams in local memory on a per byte basis. All of the source data streams and the parity data stream can be up to 16 MB long.
2. In an **XOR operation with P+Q RAID6**, the ADMA generates a parity data stream, P, that is comprised of the XOR of up to 16 distinct data streams in local memory on a per byte basis. In addition, the ADMA generates a data stream, Q, whereby the source data streams have each been Galois field multiplied followed by an XOR of the resultant data streams on a per byte basis. All of the source data streams, the P data stream, and the Q data stream can be up to 16 MB long.
3. In a **Dual XOR operation**, the ADMA generates two parity data streams that is comprised of the XOR of two source data streams, a Horizontal data stream, and a Diagonal data stream on a per byte basis. All four data streams may be up to 16 MB long.
4. In a **P+Q Update operation**, the ADMA generates updated P and Q check data streams. The updated P check data stream is comprised of the XOR of two source data streams and a P check data stream on a per byte basis. The updated Q check data stream is comprised of the XOR of the same two source data streams that have been Galois field multiplied, and a Q check data stream on a per byte basis. All four data streams may be up to 16 MB long.
5. With the **Zero Result Buffer Check Operation**, the ADMA confirms that the XOR of all the bytes of the parity stripe with their associated bytes in the streams of source data results in 00H for the entire byte count. All the source data streams and the parity data stream can be up to 16 MB long. The results of the check are written back to the Descriptor Control Word in local memory.
6. With the **P+Q Zero Result Buffer Check Operation**, the ADMA confirms that the XOR of all the bytes of the parity stripe, P, with their associated bytes in the streams of source data results in 00H for the entire byte count. Also, the ADMA confirms that the XOR of all the bytes of the Q data stream, with their associated bytes in the streams of source data after they have been Galois field multiplied results in 00H for the entire byte count. All of the source data streams, the P data stream, and the Q data stream can be up to 16 MB long. The results of the check are written back to the Descriptor Control Word in local memory.
7. Perform a **Memory Block Fill** of up to 16 MB of local memory with a 32-bit constant (CRC Address/Data).
8. Perform a **CRC Generation** on up to 16 MB of a *data transfer* stream using the value at CRC Address as a seed (CRC Address/Data). Write the result back to CRC Address. It is optional to write the data transfer stream to the destination address during this operation.

The following sections describe the seven ADMA operations in detail and support per ADMA Channel.



5.7.1 Per Channel Support for ADMA Operations

The following table describes which of the seven ADMA operations are supported in each of the 3 ADMA Channels:

Table 309. Supported ADMA Operations per ADMA Channel^a

ADMA Operation	ADMA0	ADMA1	ADMA2
Data Transfer	Y	Y	Y
XOR	Y	Y	Y
XOR with P+Q RAID6	N	N	Y
Dual XOR	Y	Y	Y
P+Q Update	N	N	Y
Zero Result Buffer Check	Y	Y	Y
Zero Result Buffer Check with P+Q RAID6	N	N	Y
Memory Block Fill	Y	Y	Y
CRC Generation	Y	Y	Y

a. Y - Supported Operation; N - Operation Not Supported



5.7.2 XOR Operation

Figure 57 describes the XOR algorithm implementation. In this illustrative example, there are four blocks of source data to be XOR-ed. The intermediate result is kept by the XOR store queue in the ADMA before being written back to local memory. The source data is located at addresses 0 A000 0400H, 0 A000 0800H, 0 A000 0C00H and 0 A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application DMA as a master on the internal bus initiates a data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

Figure 57. The Bit-wise XOR Algorithm

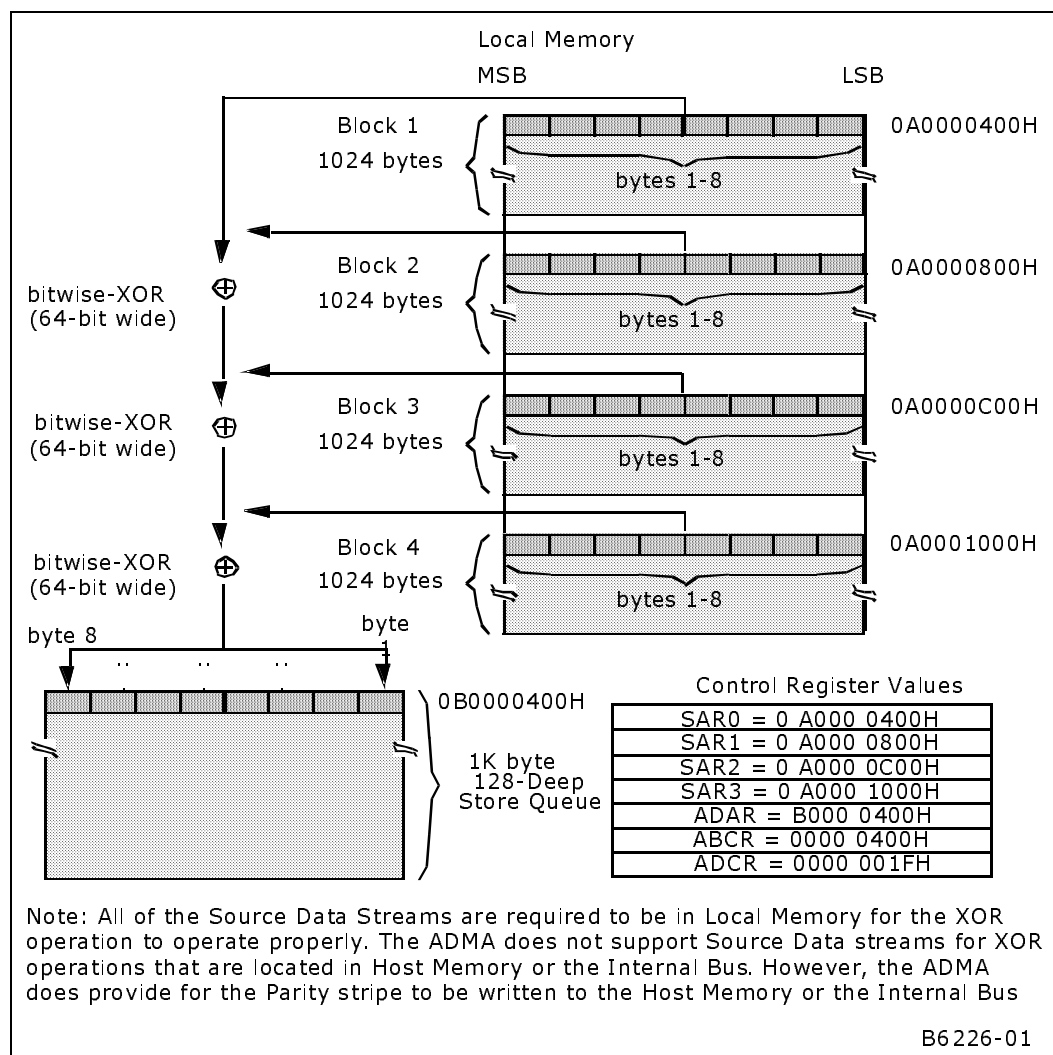
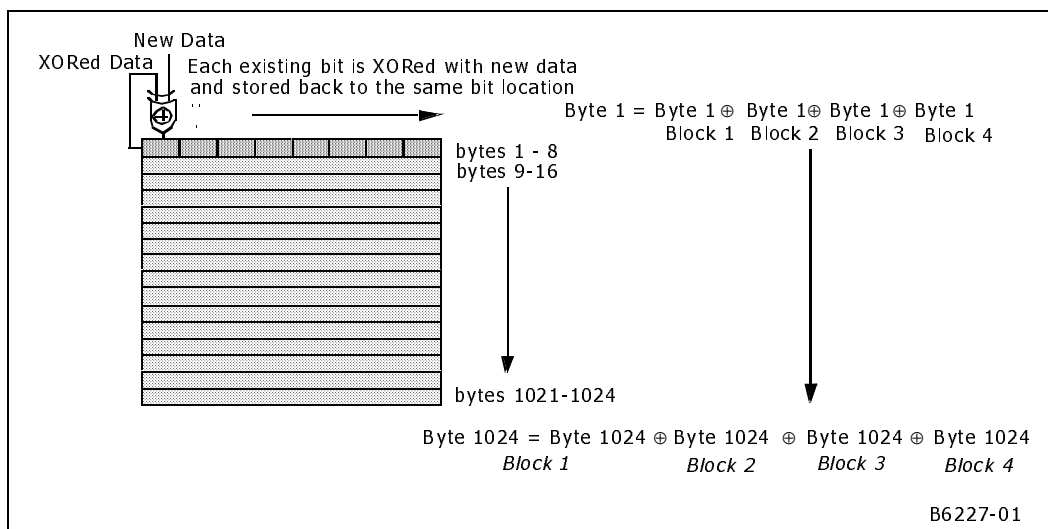


Figure 58. Hardware Assist XOR Unit





The XOR algorithm and methodology followed once a chain descriptor has been configured is detailed below:

1. The Application DMA as a master on the bus initiates data transfer from the address pointed at by the First Source Address Register (SAR0²⁰). The total number of bytes to *XOR-transfer* is specified by the Byte Count (BC) field in the chain descriptor.
 - a. This is designated as the first block of data in the current XOR operation, and the data is transferred directly to the XOR store queue. The number of bytes transferred to the XOR store queue is 1 KByte.

Note: When the Byte Count Register contains a value greater than the buffer size, the ADMA completes the *XOR-transfer* operation on the first buffer of data obtained from each Source Register (SAR0, SAR2- SAR15), then proceeds with the next buffer of data. This process is repeated until the BCR contains a zero value.

2. Application DMA transfers the first eight bytes of data from the address pointed at by the Second Source Address Register (SAR1).
3. XOR unit performs bit-wise XOR algorithm on input operands. The input operands are the first eight bytes of data read from SAR0 (bytes 1-8) which are stored in the queue and the first eight bytes of data just read from SAR1 (bytes 1-8).
4. XOR-ed result is transferred to the XOR store queue and stored in the first eight bytes (bytes 1-8) overwriting previously stored data.
5. Application DMA transfers the next eight bytes of data (bytes 9-16) from address pointed at by the Second Source Address Register (SAR1).
6. XOR unit performs the bit-wise XOR algorithm on the input operands. The input operands are the next eight bytes of data read from SAR0 (bytes 9-16 stored in the queue) and the eight bytes of data read from SAR1 in Step-5.
7. Step-5 and Step-6 (Data transfer & XOR) are repeated until all data pointed at by SAR1 is XOR-ed with the corresponding data pointed at by SAR0. The XOR store queue now contains a buffer full of XOR-ed data, the source addresses for which were specified in SAR0 and SAR1.
8. Steps 1-7 are repeated for all enabled Source Address Registers (SAR2-SAR15).
9. Once Steps 1-8 are completed, the XOR operation is complete for the first full buffer of the current chain descriptor. The data in the XOR store queue is then written to memory at the address pointed to by the Destination Lower/Upper Address Register (DLADR/DUADR) pair.
10. Byte Count is decremented by one full buffer as the data in the XOR store queue is written to memory.
11. Repeat steps 1-10 until the byte count has decremented to zero.

20.SAR0...SAR15 represent the upper/lower address register pair associated with that source (e.g., SAR0 corresponds to the SLAR0/SUAR0 register pair)

5.7.3 XOR Operation with P+Q RAID-6

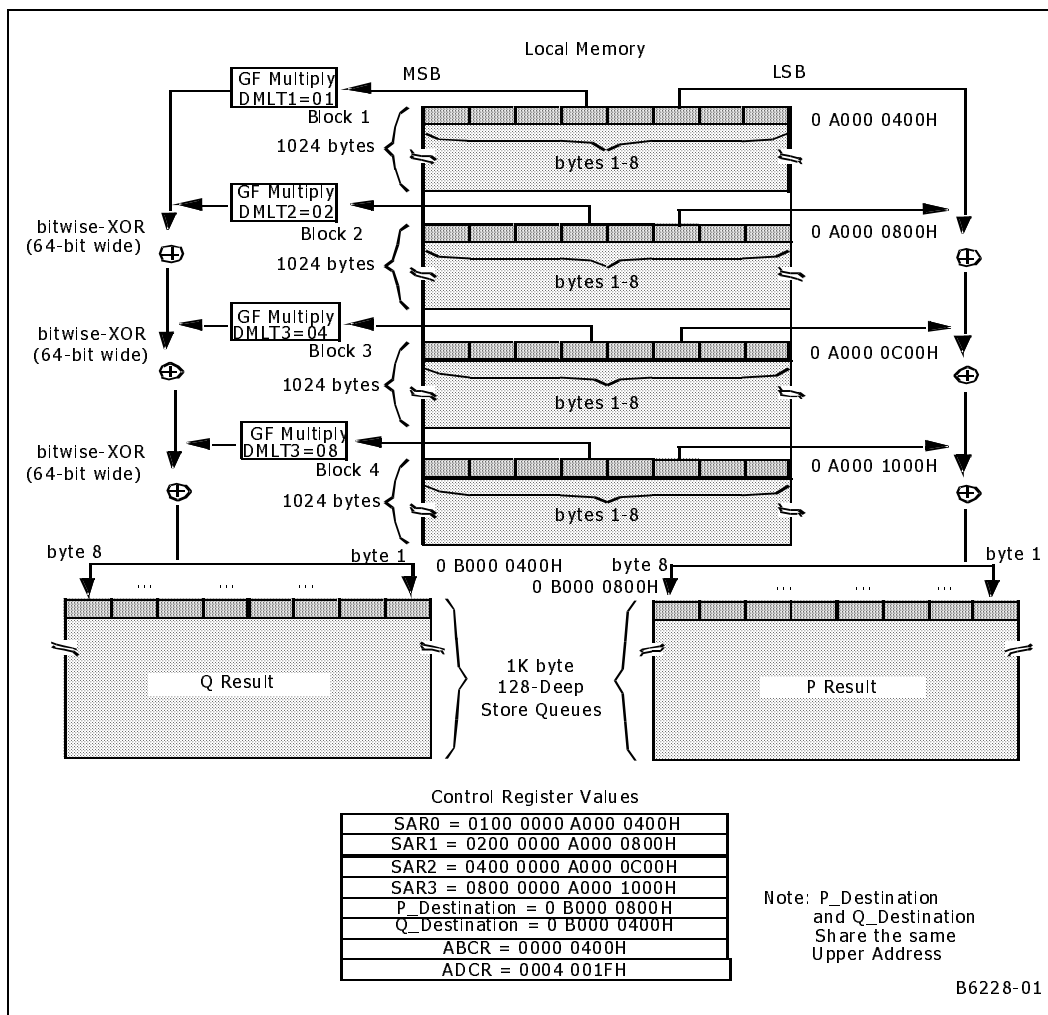
Figure 59 describes the XOR with P+Q RAID-6 implementation. In this illustrative example, there are four blocks of source data to have a P+Q RAID-6 function performed on them. The intermediate result is kept by the P and Q store queues in the ADMA before being written back to local memory. The source data is located at addresses 0 A000 0400H, 0 A000 0800H, 0 A000 0C00H and 0 A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application DMA as a master on the internal bus initiates a data transfer. The algorithm is implemented such that as data is read from local memory, the ADMA executes the XOR operation to generate P, and applies the GF Multiply function to the incoming source data followed by the XOR operation to generate Q.

Only a single descriptor is required and generates both check values. Each descriptor is processed as illustrated in Figure 59

Note: Optionally, the user may disable the generation of P while generating Q.

Figure 59. The P+Q RAID6 Mode Algorithm





The blocks for gflog (Galois Field Multiply) and gfilog (Galois Field Divide) are the Galois Field Logarithm and Inverse Logarithm transformations respectively. These transformations for 8-bit words are provided in Figure 60 and Figure 61, with the upper nibble used to index the rows, and the lower nibble used to index the columns. These are based on the primitive polynomial listed in Equation 15.

Figure 60. Galois Field Logarithm Transformation Table

Table of gflog(2^8): gflog(xy)																		
gflog(xy)		y																
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
x	0	--	00	01	19	02	32	1a	c6	03	df	33	ee	1b	68	c7	4b	
	1	04	64	e0	0e	34	8d	ef	81	1c	c1	69	f8	c8	08	4c	71	
	2	05	8a	65	2f	e1	24	0f	21	35	93	8e	da	f0	12	82	45	
	3	1d	b5	c2	7d	6a	27	f9	b9	c9	9a	09	78	4d	e4	72	a6	
	4	06	bf	8b	62	66	dd	30	fd	e2	98	25	b3	10	91	22	88	
	5	36	d0	94	ce	8f	96	db	bd	f1	d2	13	5c	83	38	46	40	
	6	1e	42	b6	ac	a3	c3	48	7e	6e	6b	3a	28	54	fa	85	ba	3d
	7	ca	5e	9b	9f	0a	15	79	2b	4e	d4	e5	ac	73	f3	a7	57	
	8	07	70	c0	f7	8c	80	63	0d	67	4a	de	ed	31	c5	fe	18	
	9	e3	a5	99	77	26	b8	b4	7c	11	44	92	d9	23	20	89	2e	
	a	37	3f	d1	5b	95	bc	cf	cd	90	87	97	b2	dc	fc	be	61	
	b	f2	56	d3	ab	14	2a	5d	9e	84	3c	39	53	47	6d	41	a2	
	c	1f	2d	43	d8	b7	7b	a4	76	c4	17	49	ec	7f	0c	6f	f6	
	d	6c	a1	3b	52	29	9d	55	aa	fb	60	86	b1	bb	cc	3e	5a	
	e	cb	59	5f	b0	9c	a9	a0	51	0b	f5	16	eb	7a	75	2c	d7	
	f	4f	ae	d5	e9	e6	e7	ad	e8	74	d6	f4	ea	a8	50	58	af	

Figure 61. Galois Field Inverse Logarithm Transformation table

Table of gfilog(2^8): gfilog(xy)																	
gfilog(xy)		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	01	02	04	08	10	20	40	80	1d	3a	74	e8	cd	87	13	26
	1	4c	98	2d	5a	b4	75	ea	c9	8f	03	06	0c	18	30	60	c0
	2	9d	27	4e	9c	25	4a	94	35	6a	d4	b5	77	ee	c1	9f	23
	3	46	8c	05	0a	14	28	50	a0	5d	ba	69	d2	b9	6f	de	a1
	4	5f	be	61	c2	99	2f	5e	bc	65	ca	89	0f	1e	3c	78	f0
	5	fd	e7	d3	bb	6b	d6	b1	7f	fe	e1	df	a3	5b	b6	71	e2
	6	d9	af	43	86	11	22	44	88	0d	1a	34	68	d0	bd	67	ce
	7	81	1f	3e	7c	f8	ed	c7	93	3b	76	ec	c5	97	33	66	cc
	8	85	17	2e	5c	b8	6d	da	a9	4f	9e	21	42	84	15	2a	54
	9	a8	4d	9a	29	52	a4	55	aa	49	92	39	72	e4	d5	b7	73
	a	e6	d1	bf	63	c6	91	3f	7e	fc	e5	d7	b3	7b	f6	f1	ff
	b	e3	db	ab	4b	96	31	62	c4	95	37	6e	dc	a5	57	ae	41
	c	82	19	32	64	c8	8d	07	0e	1c	38	70	e0	dd	a7	53	a6
	d	51	a2	59	b2	79	f2	f9	ef	c3	9b	2b	56	ac	45	8a	09
	e	12	24	48	90	3d	7a	f4	f5	f7	f3	fb	eb	cb	8b	0b	16
	f	2c	58	b0	7d	fa	e9	cf	83	1b	36	6c	d8	ad	47	8e	--

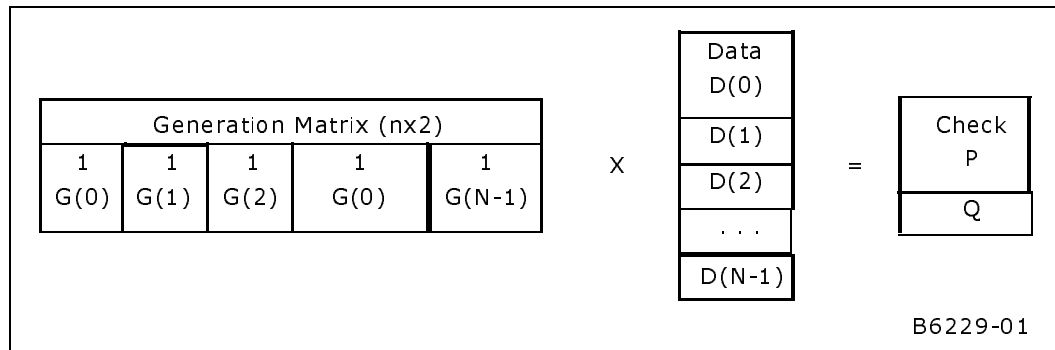
Equation 15. Galois Field Primitive Polynomial (0x11D)

$$x^8 + x^4 + x^3 + x^2 + 1$$

The P+Q RAID-6 Generation matrix contains the coefficients used in calculating the P and Q check values, based on the Galois Field Matrix multiplication provided in Figure 62. Where P is the simple XOR of the source data (The result of the GF Multiply is equal to the source data with coefficient equal to 1). The Q generation coefficients (G(0), G(1), G(2),...G(N-1)), are computed from the Vandermond matrix based on the array data disk count N.

The P and Q check values are computed by the ADMA using a single descriptor. The P+Q descriptor generates the "P" check value, where the GF Multiplier byte values are equal to 1 (first row of generation matrix). Concurrently, the ADMA generates the "Q" check value, where the GF Multiplier byte values are G(0), G(1), G(2),...G(N-1) (second row of generation matrix).

Figure 62. P+Q RAID-6 Generation Equation





5.7.4 Dual XOR Operation

Figure 63 describes the Dual XOR algorithm implementation. The Dual XOR operation generates new versions of the Horizontal and Diagonal parity check data blocks when a single block write occurs to a data storage subsystem that is protected by two-dimensional XOR RAID6. In this illustrative example, there are four blocks of source data to be XOR-ed. The intermediate results are kept by the XOR store queue in the ADMA before being written back to local memory. The source data is located at addresses 0 A000 0400H, 0 A000 0800H, 0 A000 0C00H and 0 A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application DMA as a master on the internal bus through its' dedicated DDR SDRAM MCU port initiates a data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

Figure 63. The Bit-wise Dual XOR Algorithm

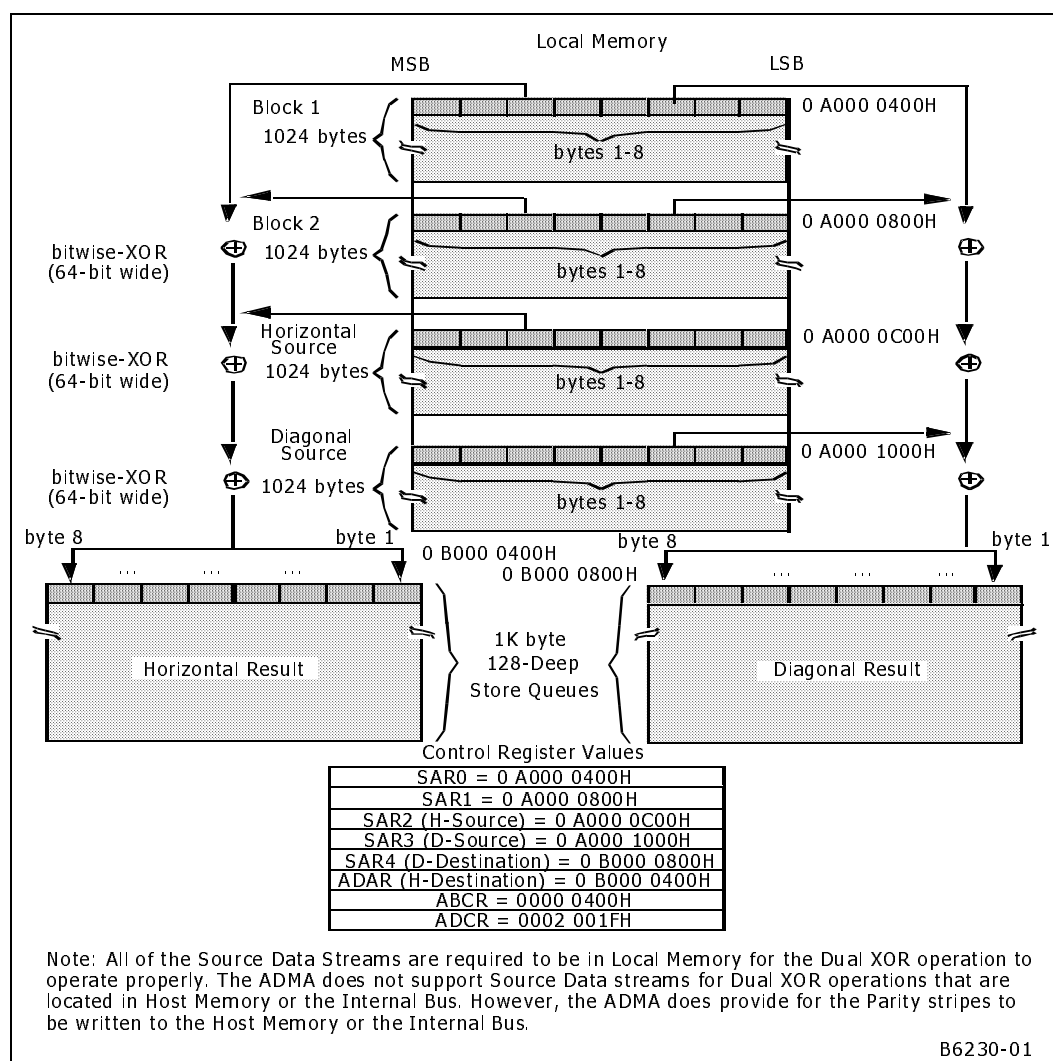
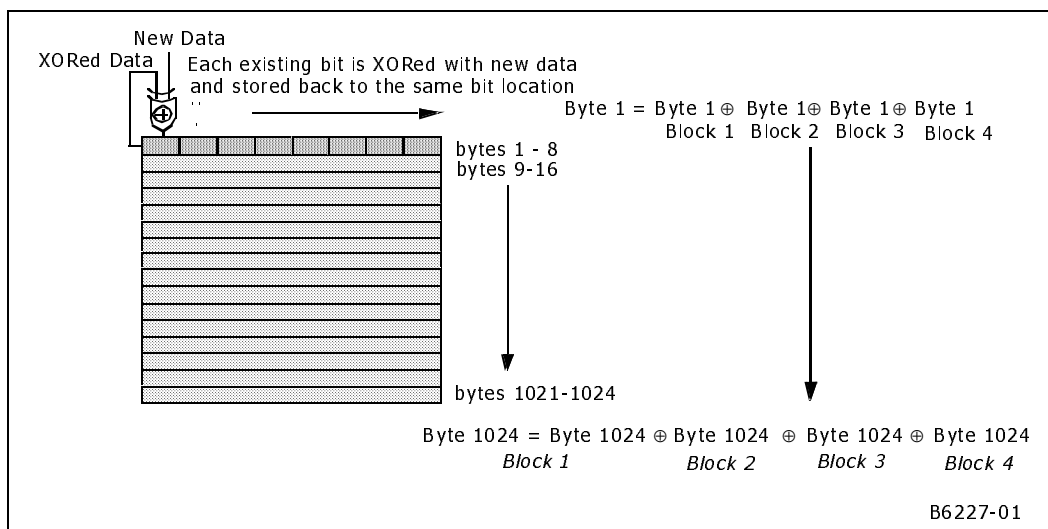


Figure 64. Hardware Assist XOR Unit





The Dual XOR algorithm and methodology followed once a chain descriptor has been configured is detailed below:

1. The Application DMA as a master on the bus initiates data transfer from the address pointed at by the First Source Address Register (SAR0²¹). The total number of bytes to *Dual-XOR-transfer* is specified by the Byte Count (BC) field in the chain descriptor.
 - a. This is designated as the first block of data in the current Dual XOR operation, and the data is transferred directly to the XOR store queue. The number of bytes transferred to the XOR store queue is 1 KByte.

Note: When the Byte Count Register contains a value greater than the buffer size, the ADMA completes the *Dual-XOR-transfer* operation on the first buffer of data obtained from each Source Register (SAR0, SAR1, Horizontal Source Data (SAR2), Diagonal Source Data (SAR3)), then proceeds with the next buffer of data. This process is repeated until the BCR contains a zero value.

2. The Application DMA transfers the first eight bytes of data from the address pointed at by the Second Source Address Register (SAR1).
3. The XOR unit performs the bit-wise XOR algorithm on the input operands. The input operands are the first eight bytes of data read from SAR0 (bytes 1-8) which are stored in the queue and the first eight bytes of data just read from SAR1 (bytes 1-8).
4. The XOR-ed result is transferred to the XOR store queue and stored in the first eight bytes (bytes 1-8) overwriting previously stored data.
5. The Application DMA transfers the next eight bytes of data (bytes 9-16) from address pointed at by the Second Source Address Register (SAR1).
6. The XOR unit performs the bit-wise XOR algorithm on the input operands. The input operands are the next eight bytes of data read from SAR0 (bytes 9-16 stored in the queue) and the eight bytes of data read from SAR1 in Step-5.
7. Step-5 and Step-6 (Data transfer & XOR) are repeated until all data pointed at by SAR1 is XOR-ed with the corresponding data pointed at by SAR0. The XOR store queue now contains a buffer full of XOR-ed data, the source addresses for which were specified in SAR0 and SAR1.
8. Steps 2-7 are repeated for the Horizontal Source. However, the XOR results are written to the Destination store queue rather than being written back to the XOR store queue as in steps 2-7.
9. The data in the Destination store queue is then written to memory at the address pointed to by the Horizontal Destination Address (DLADR/DUADR).
10. Steps 2-7 are repeated for the Diagonal Source. However, the XOR results are written to the Destination store queue rather than being written back to the XOR store queue as in steps 2-7.
11. The data in the Destination store queue is then written to memory at the address pointed to by the Diagonal Destination Address (SAR4).
12. Once Steps 1-11 are completed, the Dual XOR operation is complete for the first full buffer of the current chain descriptor.
13. The Byte Count is decremented by one full buffer as the Diagonal Parity result in the Destination store queue is written to memory.
14. Repeat steps 1-13 until the byte count has decremented to zero.

21. SAR0...SAR3 represent the upper/lower address register pair associated with that source (e.g., SAR0 corresponds to the SLAR0/SUAR0 register pair)



5.7.5 P+Q Update Operation

The P+Q Update operation generates new versions of the P and Q check data blocks when a single block write occurs to a data storage subsystem that is protected by P+Q RAID6. [Figure 65, “The P+Q Update Algorithm” on page 501](#) describes the P+Q Update operation implementation. In this illustrative example, an old data block (SAR0), a new data block (SAR1), an old P check data block (SAR2), and an old Q check data block (SAR3) are used to generate a new P check data block and a new Q check data block. The intermediate result is kept in the ADMA P and Q result buffers before being written back to local memory. The source data is located at addresses 0 A000 0400H, 0 A000 0800H, 0 A000 0C00H and 0 A000 1000H respectively.

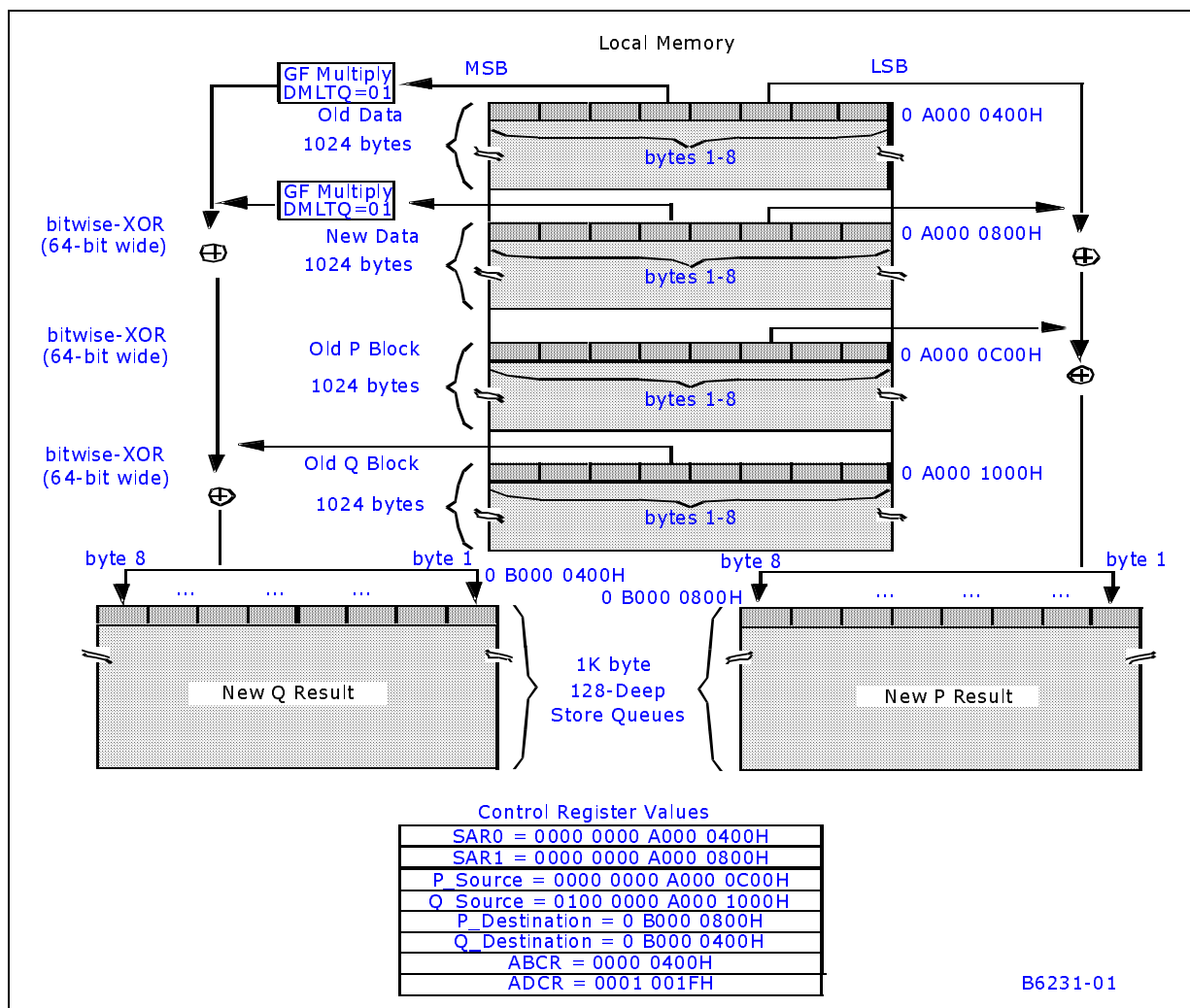
All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application DMA as a master on the internal bus and through its' dedicated DDR SDRAM MCU port initiates the data transfer. The algorithm is implemented such that as data is read from local memory, the ADMA executes the XOR operation to generate new P, and applies the GF Multiply function to the incoming source data followed by the XOR operation to generate new Q.

Only a single descriptor is required and generates both check values. Each descriptor is processed as illustrated in [Figure 65 on page 501](#)

Note: Optionally, the user may disable the generation of new P while generating new Q.



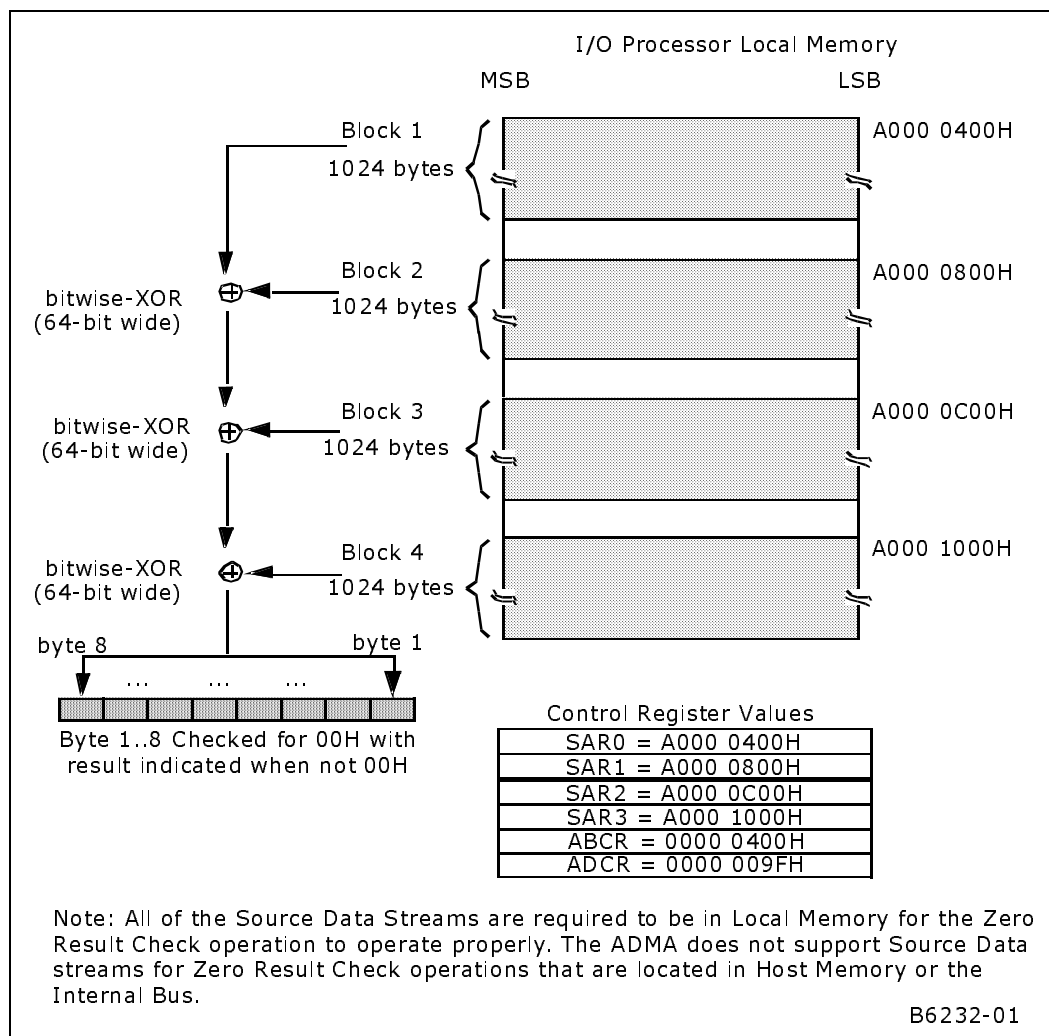
Figure 65. The P+Q Update Algorithm



5.7.6 Zero Result Buffer Check

The ADMA can be used to verify parity across memory blocks specified by the SARx registers. As with XOR operations, descriptors are used to specify the memory blocks on which the ADMA performs the Zero Result Buffer Check. Figure 66 illustrates a Zero Result Buffer Check performed by the ADMA. After processing all source data and depending on the setting of the Status Write Back Enable in the ADCR, the ADMA either updates the Result Buffer Not Zero and Transfer Complete bits of the fourth word of the descriptor (ABCR) or signals an interrupt to the Intel XScale® processor and suspend ADMA operation.

Figure 66. An Example of Zero Result Buffer Check





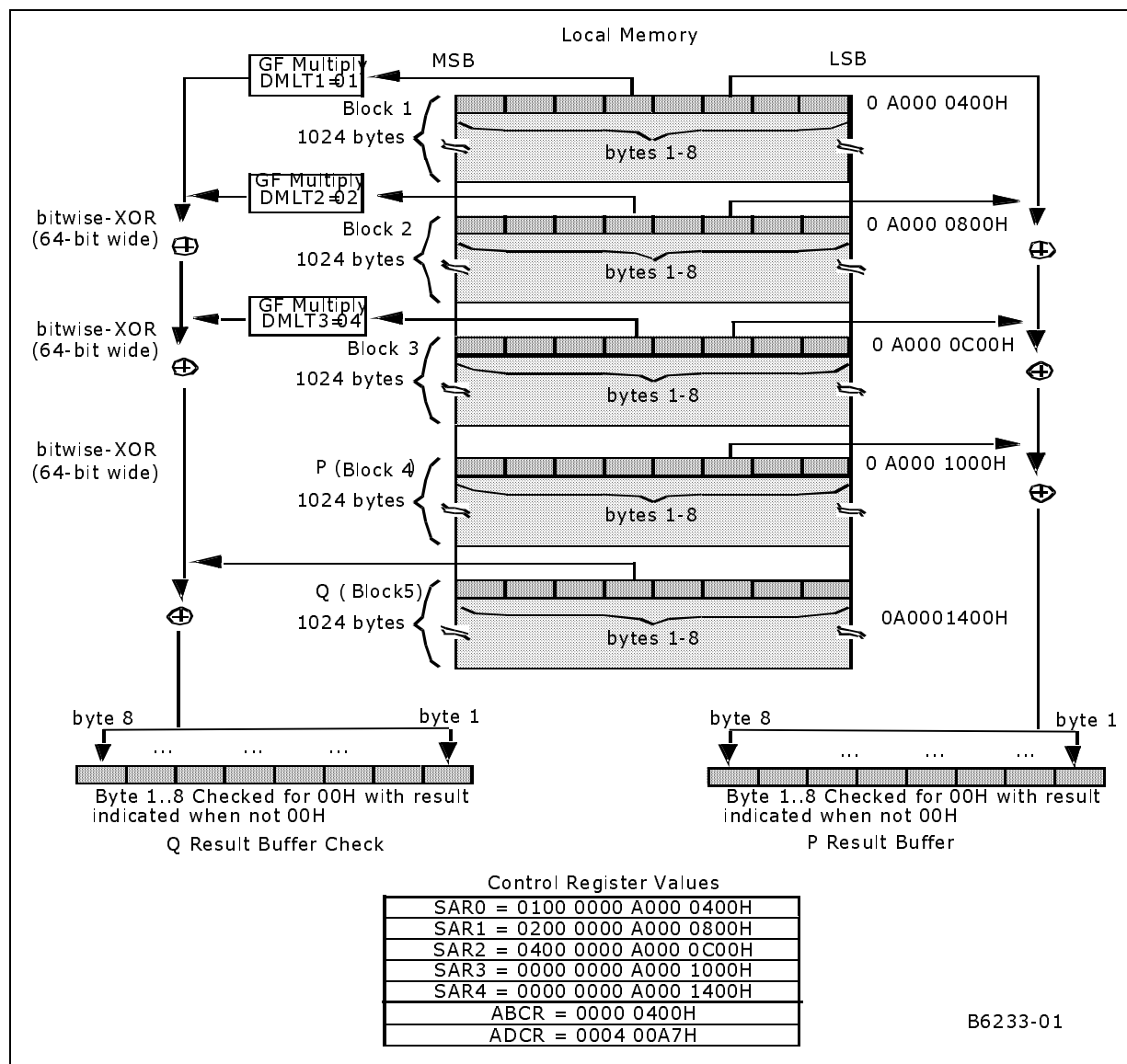
5.7.7 P+Q Zero Result Buffer Check

The ADMA can be used to verify check values for P+Q RAID-6 implementations. As with P+Q operations, descriptors are used to specify the memory blocks on which the ADMA performs the P+Q Zero Result Buffer Check. [Figure 67 on page 504](#) illustrates a P+Q Zero Result Buffer Check performed by the ADMA. After processing all source data and depending on the setting of the Status Write Back Enable in the ADCR, the ADMA either updates the P Result Buffer Not Zero, Q Result Buffer Not Zero, and Transfer Complete bits of the fourth word of the descriptor (ABCR) or signals an interrupt to the Intel XScale® processor and suspend ADMA operation.

This operation can perform a consistency verification of the P and Q check data blocks associated with up to 14 data blocks. Even though there are 16 Source Addresses available, the next to last source address, and the last source address are reserved for the associated P and Q check data blocks upon which the consistency verification is performed. For example, in [Figure 67 on page 504](#), five source addresses are programmed into the ADCR where SAR0-SAR2 represent three data blocks, SAR3 represents the P check data block, and SAR4 represents the Q check data block.

Note: By setting the 'P Transfer Disable' bit in the ADCRx, the user may disable the Zero Result Buffer Check of P while performing the Zero Result Buffer Check on Q.

Note: The valid minimum number of sources to use in order to perform a P+Q Zero Result Buffer Check is four data blocks. The four sources include the two source data blocks, the P data block, and the Q data block.

Figure 67. Zero Result Buffer Check with P+Q RAID6


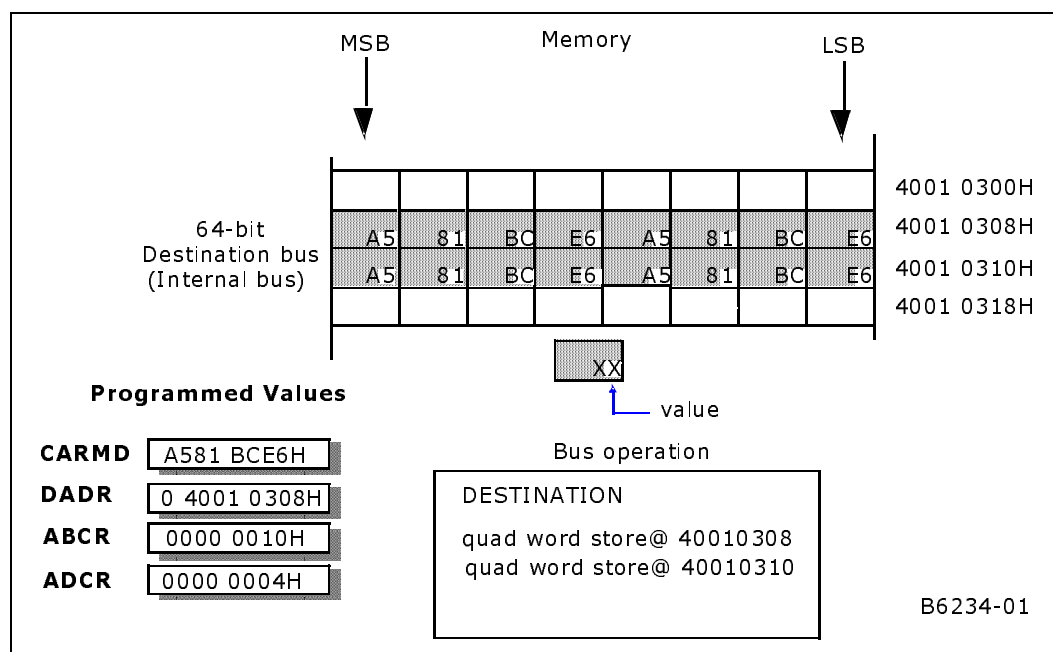


5.7.8 Memory Block Fill Operation

The ADMA can be used to write a constant value to any memory block. As with XOR operations, descriptors are used to specify the memory blocks to which the ADMA writes the data contained in the CRC Address / Memory Block Fill Data Register (CARMDQ). All memory block fill operations are controlled by chain descriptors located in memory. Figure 68 illustrates a Block Fill Operation to an arbitrary destination address.

Note: The Destination Address is used to designate the starting address for the memory block fill.

Figure 68. Example of a Memory Block Fill Operation





5.8 CRC Generation

When enabled, the ADMA generates a 32-bit CRC based on the programmed data stream and a 32-bit seed. The CRC engine uses the CRC-32C algorithm required by the iSCSI* Specification.

5.8.1 CRC Mode Configuration and Operation

In addition to the normal ADMA Descriptor configuration, the following additional steps are required to configure the CRC engine:

1. When writing out the descriptor, bit 9 of the DC is set to enable CRC generation. For a single descriptor CRC operation or the first descriptor of a multiple descriptor CRC operation, bit 11 of the DC is cleared to enable the CRC seed fetch. See [Section 5.8.1.1](#) for details of how to properly use bit 11 and bit 9 of the DC to chain a CRC operation across multiple descriptors.
2. Word 2 of the descriptor is written with the 32-bit aligned CRC Address.

When a descriptor is configured to generate CRC, the ADMA performs the following steps in addition to the Data Transfer:

Note: When Word 3 (Byte Count) is set to 0, then the CRC seed value is not affected and the ADMA advances to the next descriptor.

1. Prior to the start of the Data Transfer, the ADMA loads the 32-bit seed value from the CRC Address into an internal CRC Shift Register.
2. The ADMA accumulates the 32-bit CRC value in an internal register as the Data Transfer is occurring. Specifically, this CRC internal register is recirculated through the CRC-32C algorithm using the current contents of this register and the most recent data transferring through the ADMA.
 - a. $\text{CRC Internal Register (n)} = \text{CRC-32C}(\text{CRC Shift Register (n-1)}, \text{Data(n)})$.
3. Following the completion of the Data Transfer, the ADMA transposes this internal register's contents into a CRC value and write it to CRC Address.
4. Finally, the ADMA writes the Byte Count Register back to the descriptor with the Transfer Complete bit set.

Note: See [Figure 70, "Calculation of 32-bit CRC for iSCSI PDU" on page 508](#) for details on the implementation.

Note: ADCR bit 10 may be used to disable the Data Transfer during a CRC calculation. This provides the ability to generate CRC on a source data block without writing a destination data block.



5.8.1.1 Chaining of CRC Calculation Across Multiple Descriptors

Using the CRC Address (Word 2 of the descriptor) and the CRC Seed Fetch Disable (bit 11 of the DC), the CRC calculation can be chained across multiple descriptors. Since the CRC-32C is associated with an iSCSI PDU (Protocol Data Unit), this may be necessary as the network packet data associated with a particular PDU can be presented as multiple, non-contiguous fragments in the local memory. For example, [Table 310 on page 507](#) illustrates the use of three descriptors to complete a CRC-32C calculation for a PDU that has three fragments.

Note: The set of descriptors used to complete a multiple descriptor CRC operation is required to be in the descriptor chain in the correct data order for the CRC-32C to be calculated properly. However, in the same descriptor chain, there may be intervening non-CRC enabled descriptors or CRC enabled descriptors associated with another PDU (e.g., CRC Address is different). For example, Descriptor 3 in [Table 310 on page 507](#).

Table 310. Multiple Descriptor Usage for CRC Calculation of a 3 Fragment PDU

Order of Descriptors in Chain	CRC Enable (bit 9 of DC)	CRC Seed Fetch Disable (bit 11 of DC) ^a	Initial Value at CRC Address	Final Value at CRC Address	Comments
1	1	0	FFFF FFFFh	CRC for Fragment 1 ^b	First Fragment Seeds the CRC Calculation with Value in Memory
2	1	1	CRC for Fragment 1	CRC for Fragment 2	Second Fragment Descriptor uses CRC Value of First Fragment in Memory to continue the operation
3	0	x	Not Applicable	Not Applicable	non-CRC Enabled Descriptor
4	1	1	CRC for Fragment 2	Final CRC for 3 Fragment PDU	Third Fragment Descriptor uses CRC Value of Second Fragment in Memory to complete the operation

- a. When CRC Seed Fetch Disable is cleared, the 32-bit value at CRC Address is loaded into the CRC Internal Register. When CRC Seed Fetch Disable is set, the inverted and bit-wise (per byte) transposed value at CRC Address is loaded into the CRC Internal Register. Please see [Figure 70, "Calculation of 32-bit CRC for iSCSI PDU" on page 508](#).
- b. When the initial byte count (Descriptor word 3) equals 0, then there would be no effect on the CRC and the Final Value at CRC Address would still be FFFF_FFFFh.

5.8.2 CRC-32C Algorithm

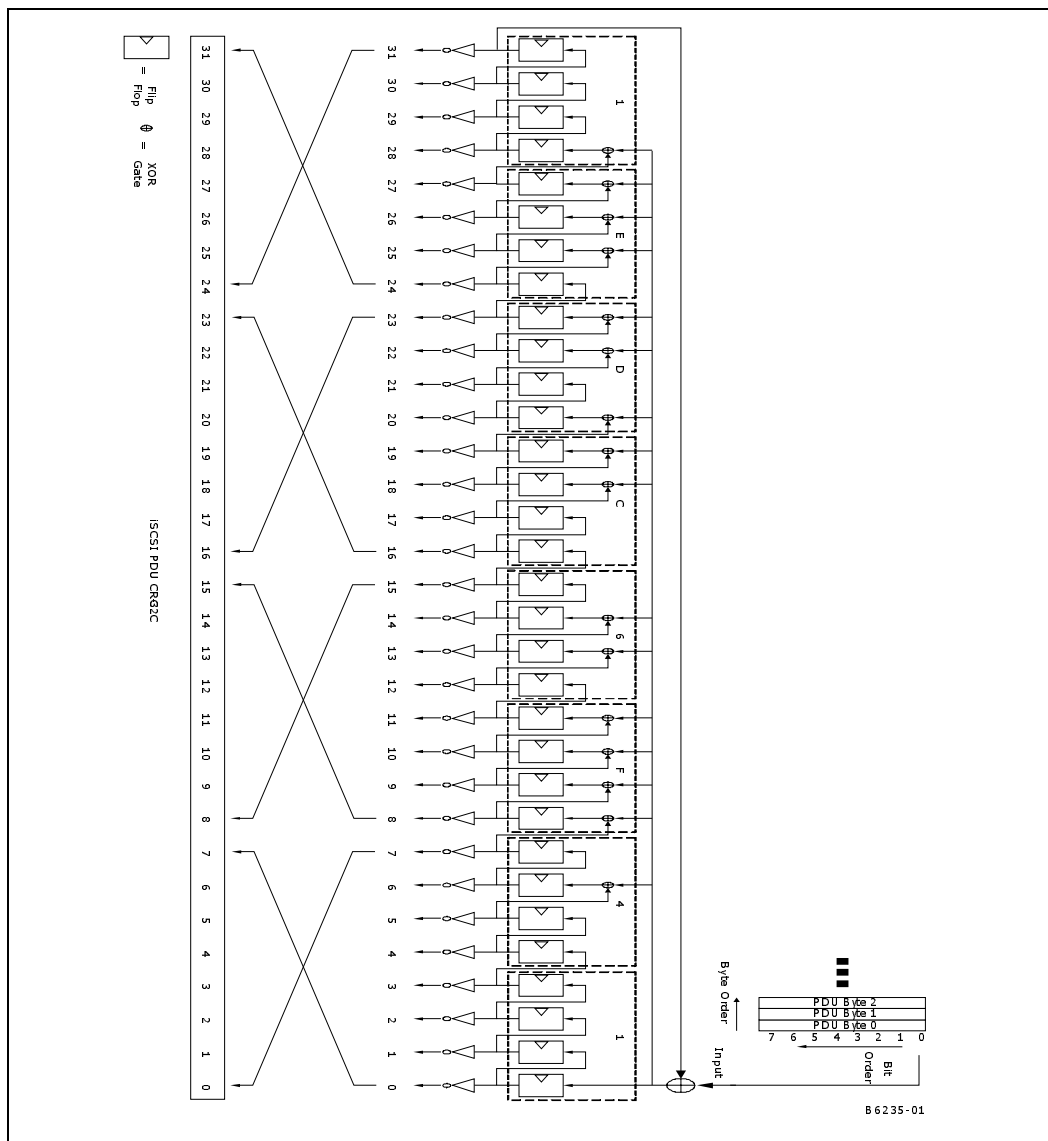
The CRC Engine generates a 32-bit CRC of the programmed data stream using the CRC-32C generator polynomial required by the iSCSI* Specification, (1EDC6F41).

The incoming data stream is shifted and recirculated through the CRC polynomial using a method similar to that illustrated in Figure 70, "Calculation of 32-bit CRC for iSCSI PDU" on page 508. The initial seed for this operation is required to be FFFF FFFFh which is equivalent to the initial value of the CRC internal register in Figure 70.

Figure 69. CRC-32C Generator Polynomial

$$1 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{18} + x^{19} + x^{20} + x^{22} + x^{23} + x^{25} + x^{26}$$

Figure 70. Calculation of 32-bit CRC for iSCSI PDU

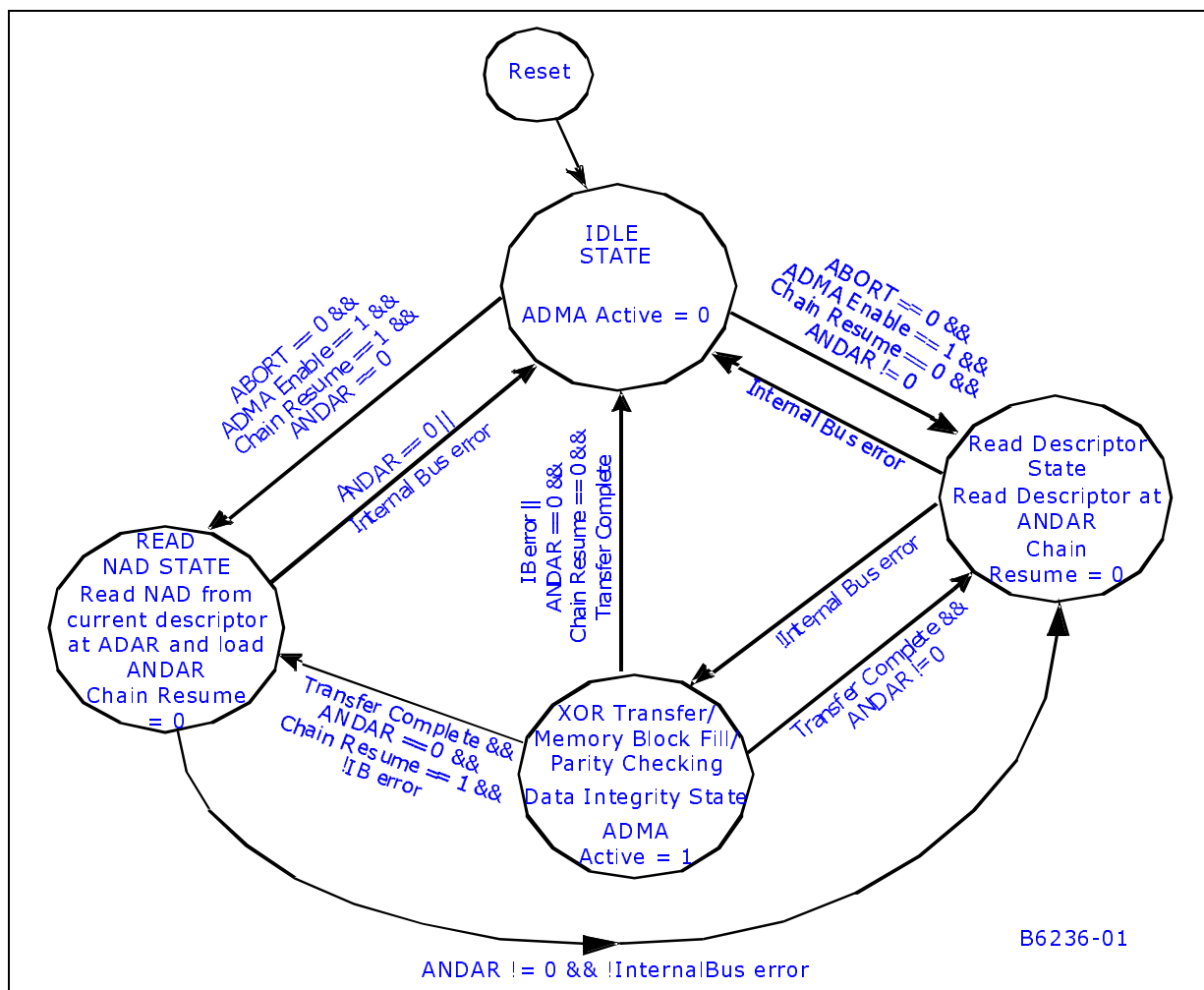




5.9 Programming Model State Diagram

The ADMA programming model diagram is shown in Figure 71. Error condition states are not shown.

Figure 71. Application DMA Programming Model State Diagram





5.10 Application DMA Priority

The internal bus arbitration logic determines which internal bus master has access to the 81341 and 81342 internal bus. The Application DMA has an independent Bus Request/Grant signal pair to the internal bus arbitration logic. [Chapter 6.0, “System Controller \(SC\) and Internal Bus Bridge”](#) describes in detail the priority scheme between all of the bus masters on the internal bus.



5.11 Programming the Application DMA

The software for initiating a transfer using the Application DMA falls into the following categories:

- ADMA initialization
- Start Transfer
- Suspend ADMA

An example for each category is shown in the following sections as pseudo code flow.

5.11.1 Application DMA Initialization

The ADMA is designed to have independent control of the interrupts, enables, and control. The initialization consists of virtually no overhead as shown in [Figure 72](#).

Figure 72. Pseudo Code: Application DMA Initialization

```
ACCR = 0x0000 0000 ; Disable the Application DMA
Call setup_adma
```

5.11.2 Start Transfer

The ADMA control register provides independent control each time the ADMA is configured. This provides the greatest flexibility to the applications programmer. The example shown in [Figure 73](#) describes the pseudo code for initiating an XOR operation with the ADMA for four sources.

Figure 73. Pseudo Code: XOR Transfer Operation

```
; Set up descriptor in Intel XScale® microarchitecture local memory at address d
d.nda = 0          /* No chaining */

d.SAR0 = 0x0 A000 0400/* Source address of Data Block 1      */
d.SAR1 = 0x0 A000 0800/* Source address of Data Block 2      */
d.SAR2 = 0x0 A000 0C00/* Source address of Data Block 3      */
d.SAR3 = 0x0 A000 1000/* Source address of Data Block 4      */
d.DADR = 0x0 B000 0100/* Destination address of XOR-ed data */
d.ABCR = 1024        /* Byte Count of 1024 */
d.ADCR = 0x0000 001F/* XOR operation on 4 Source Blocks */

/* Store the result & interrupt processor */

; Check for Inactive ADMA & no pending interrupts
if (ACSR != 0) exit/* If ADMA is not ready, exit */

; Start Operation
ANDAR = &d          ; Set up descriptor address
ACCR = 0x00000001   ; Set ADMA Enable bit
```



5.11.3 Suspend Application DMA

The Application DMA unit provides the ability to suspend the current state without losing status information. The ADMA resumes without requiring application software to save the current configuration. [Figure 74](#) describes pseudo-code for suspending ongoing operation and restarting.

Figure 74. Pseudo Code: Suspend Application DMA

```
ACCR = 0x0000 0000 ; Suspend ongoing ADMA transfer  
  
;Suspend Application DMA  
  
ACCR = 0x0000 0001 ; Resume ADMA transfer
```



5.12 Internal Interface Parity Checking and Generation

The south internal bus interface of the ADMA supports byte-wise parity on both the address bus and data bus. The ADMA can act as both an initiator and a target on the south internal bus. The ADMA acts as a target when its memory-mapped registers are being accessed from the south internal bus. The ADMA acts as an initiator when it fetches descriptors or while transferring data. Note that the ADMA fetches descriptors via the south internal bus, and the descriptors are normally located in the DDR memory.

As a target when the ADMA memory-mapped registers are being accessed, the ADMA behavior is described as follows:

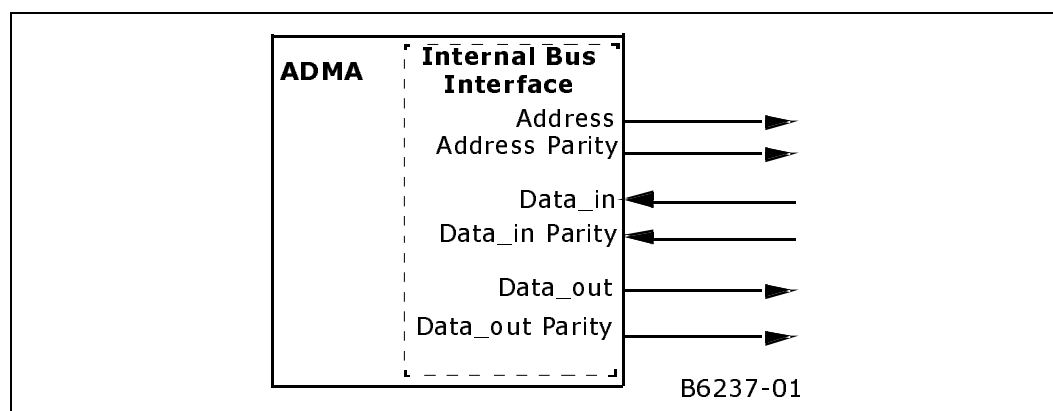
- For a write request, the ADMA internal bus interface checks the address parity during the address request phase, asserts the internal bus error signal when it detects an address parity error. The address parity error is logged by the System Controller. Refer to the [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#). However, during the data phase of a write request, the ADMA does **not** check the data parity.
- For a write request, the ADMA internal bus interface checks the address parity during the address request phase, asserts the internal bus error signal when it detects an address parity error. The address parity error is logged by the System Controller. Refer to the [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#). When returning data during read completions, the ADMA simply forwards the data and parity generated by the DMCU on the south internal bus.

As an initiator the ADMA behavior is as follows:

- To issue a read request on the internal bus, the ADMA generates address parity before driving the address on the internal bus. When receiving data during a read completion, the ADMA forwards the data and parity to the DMCU. For example, the ADMA does **not** verify the data parity. However, when receiving data during a read completion for a descriptor fetch, the ADMA verifies the parity on the descriptor data. When a data parity error is detected, the ADMA flags the parity error.
- To issue a write request on the internal bus, the ADMA generates the address parity before driving the address on the south internal bus. For data, the ADMA does **not** generate data parity, it simply forwards the data and parity as received from the DMCU.

Note: The Memory Port Interface of the ADMA does not verify or generate parity.

Figure 75. Logical Data Access Paths with Parity Protection





5.12.1 Address and Data Parity Generation

5.12.1.1 Address Parity Generation

As an initiator, the ADMA internal bus interface generates and drives even parity on the internal address bus for either a read or a write request. When the target of the ADMA request detects an address parity error, the target discards the request by asserting the internal bus error signal. However, the address parity error is logged by the System Controller. Refer to the [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#).

Note: The direct memory port does not support address parity.



5.12.1.2 Data Parity Generation

As a target during a memory-mapped register read access, the ADMA internal bus interface generates and drives data parity when returning data for memory-mapped register read requests.

As an initiator, the ADMA does **not** generate data parity. When receiving data from the internal bus, the ADMA simply forwards the data and its parity to the DMCU as received from the internal bus. When receiving data from the DMCU, the ADMA simply forwards the data and its parity to the internal bus as received from the DMCU.

Table 311 and Table 312 respectively list the address and data bytes that are used for the address and data parity calculation. The parity bits are calculated by bit XORing the data bits shown in Table 311. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is calculated as follows:

Equation 16. $D\_PARITY0 = D[0] \text{ XOR } D[1] \text{ XOR } D[2] \text{ XOR } D[3] \text{ XOR } D[4] \text{ XOR } D[5] \text{ XOR } D[6] \text{ XOR } D[7]$

Table 311. Address Parity Checking/Generation

Address Parity Bit	Address Byte
A_PARITY4	A[35:32]
A_PARITY3	A[31:24]
A_PARITY2	A[23:16]
A_PARITY1	A[15:8]
A_PARITY0	A[7:0]

Table 312. Data Parity Checking/Generation

Data Parity Bit	Data Byte	Byte Enable
D_PARITY15	D[127:120]	BE15
D_PARITY14	D[119:112]	BE14
D_PARITY13	D[111:104]	BE13
D_PARITY12	D[103:96]	BE12
D_PARITY11	D[95:88]	BE11
D_PARITY10	D[87:80]	BE10
D_PARITY9	D[79:72]	BE9
D_PARITY8	D[71:64]	BE8
D_PARITY7	D[63:56]	BE7
D_PARITY6	D[55:48]	BE6
D_PARITY5	D[47:40]	BE5
D_PARITY4	D[39:32]	BE4
D_PARITY3	D[31:24]	BE3
D_PARITY2	D[23:16]	BE2
D_PARITY1	D[15:8]	BE1
D_PARITY0	D[7:0]	BE0



5.12.2 Address and Data Parity Checking

5.12.2.1 Address Parity Checking

As a target, the ADMA internal bus interface checks for address parity on a read or a write request that it claims.

On the internal bus, address parity is checked on the address bus on a byte by byte basis. The address parity bits are checked by first calculating the parity bits on the incoming address bytes shown in [Table 311](#) and verifying the results against the corresponding incoming address parity signals. As an example, the parity calculation for the lowest order byte of the address bus A[7:0] is carried as follows:

Equation 17. ADDRESS_PARITY_RESULT = A_PARITY0 XOR A[0] XOR A[1] XOR A[2] XOR A[3] XOR A[4] XOR A[5] XOR A[6] XOR A[7]

The parity logic uses the following algorithm. This algorithm logs the error when an error is detected.

```
check address_parity_result
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```




5.12.2.2 Data Parity Checking

As an initiator, the ADMA internal bus interface checks for data parity while receiving read completion data from the internal bus only during descriptor fetches.

As a target, the ADMA internal bus does **not** check for data parity when receiving data from the internal bus due to memory-mapped register writes.

Data parity is checked on the data bus on a byte by byte basis. The data parity calculation includes parity on the data byte and the corresponding byte enable signal. The parity bits are checked by first calculating the parity bits on the incoming data bytes and the corresponding byte enable signals for each data byte shown in [Table 312](#) and verifying the results against the corresponding incoming data parity bits. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is carried as follows:

Equation 18.
$$\text{DATA\_PARITY\_RESULT} = \text{D\_PARITY0 XOR D[0] XOR D[1] XOR D[2] XOR D[3] XOR D[4] XOR D[5] XOR D[6] XOR D[7] XOR BE0}$$

A non-zero result from the above operation indicates a parity error.

The parity logic uses the following algorithm, and this algorithm logs the error when an error is detected.

```
check data_parity_result
    if parity is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

5.12.2.3 Parity Disabled

When software disables data parity, the ADMA would simply not verify data parity on descriptor fetches. Note that address parity is always enabled.

5.12.2.4 Parity Testing

The System Controller provides the ability for the programmer to test error handling software by forcing address or data parity error. Refer to the [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#) for more details.

5.13 Interrupts

ADMA can generate an interrupt to the 81341 and 81342. The *Interrupt Enable* bit in the ADMA Descriptor Control Register (ADCR.ie) determines whether the ADMA generates an interrupt upon successful, error-free completion. Error conditions described in [Section 5.14](#) also generate an interrupt. Each ADMA has three interrupt outputs connected to the Interrupt Controller Unit (see [Chapter 3, “Interrupt Controller Unit”](#)):

- ADMA x End-of-Transfer Interrupt
- ADMA x End-of-Chain Interrupt
- ADMA x Error Interrupt

where x = 0, 1, and 2 for three ADMA channels.

Once the ADMA is enabled, the ADMA loads the chain descriptor fields into the respective registers. [Table 313 on page 518](#) summarizes the status flags and conditions when interrupts are generated in the ADMA Channel Status Register (ACSR).

Table 313. ADMA Interrupt Summary

Interrupt Condition	Channel Status Register (ACSR) Flags								Interrupt Generated		
	Active	End of Transfer	End of Chain	Internal Interface Parity Error	Internal Bus Master Abort	Internal Bus Target Abort	MCU Port Abort	Zero Result Buffer Error	Zero Result Buffer Error Q	Interrupt Enabled	Interrupt Disabled
Byte count == 0 && (NDARx != NULL Resume == 1 ^a) (End of Transfer)	1	1	0	0	0	0	0	0	0	Y	N
Byte Count == 0 && NDARx == NULL (End of Chain) & resume == 0	0	X ^b	1 ^a	0	0	0	0	0	0	Y	N
Internal Interface Parity Error	0	X	X	1	0	0	0	0	0	Y	Y
Internal Bus Master Abort	0	X	X	0	1	0	0	0	0	Y	Y
Internal Bus Target Abort (Host Master Abort, Host Target Abort, Unknown Split Error, or other Internal Bus errors)	0	X	X	0	0	1	0	0	0	Y	Y
MCU Port Abort	0	X	X	0	0	0	1	0	0	Y	Y
Zero Result Buffer Error ^c Zero Result Buffer Error P	0	X	X	0	0	0	0	1	0	Y	Y
Zero Result Buffer Error Q ^c	0	X	X	0	0	0	0	0	1	Y	Y

a. During normal operation (non-error), when the Resume bit is set while the last descriptor is being executed, the ADMA re-reads the current descriptor to get the address of the appended chain descriptor. When the re-read last descriptor contains a NULL NDARx pointer, the ADMA goes inactive by also setting the End-of-Chain bit. This is a boundary condition that can occur in a scenario where one or more descriptors were appended to the end of the previous descriptor chain, but the Resume bit was not set until the last descriptor of the appended descriptor(s) was being executed.

b. “X” - Don’t Care.

c. The error status bit in the ACSR is set and an error interrupt is signaled to the Intel XScale® processor when the Status Write Back Enable bit in the ADCR is cleared. When the Status Write Back Enable in the ADCR is set, the error status bit and the Transfer Complete bit in the Transfer Status field of the ABCR are written back to memory; the error status bit in the ACSR is not set and there is no Intel XScale® processor interrupt.

Note: End-of-Transfer and End-of-Chain flags are set only when Interrupt Enable is set. When Interrupt Enable is clear, then the above flags are always set to 0. End-of-Transfer Interrupt and End of Chain Interrupt can only be reported in the ACSR when the descriptor fetch and processing completed without any reportable errors. However, multiple error conditions may occur and be reported together. Also, because the ADMA does not stop after reporting the End-of-Transfer interrupt, an error may occur before the End-of-Transfer interrupt is serviced and cleared.



5.14 Error Conditions

5.14.1 Fatal Errors

Internal Interface Parity Errors, Internal Bus Master-Abort, Internal Bus Target-Abort, and the MCU Port Abort are fatal errors that are recorded by the Application DMA. In addition, when the Status Write Back Enable in the ADCR is cleared, the Zero Result Buffer Error is recorded as a fatal error by the Application DMA.

When a fatal error occurs, the actions taken are detailed below:

- The ADMA shall cease the ongoing transfer for the current chain descriptor and clear the *Application DMA Active* flag in the ACSR.
- The ADMA does not read any new chain descriptors.
- The ADMA sets the error flag in the ADMA Status Register. For example; when an Internal Bus master-abort occurred during a transfer, the channel sets bit 5 in the ACSR.
- The ADMA signals an interrupt to the Intel XScale® processor and clears the ADMA Enable bit in the ACCR.
- The Application DMA does not restart the transfer after an error condition. It is the responsibility of the application software to reconfigure the ADMA to complete any remaining transfers.

Note: Target-aborts during ADMA reads result either from multi-bit ECC errors that are recorded by the MCU or target abort conditions from the host I/O interface which are recorded by the host I/O interface. The ADMA needs to be re-enabled by writing a 1 to the ADMA Enable bit before initiating a new operation.

5.14.2 Non-Fatal Errors

When the Status Write Back Enable in the ADCR is set, the Zero Result Buffer Error is handled as a non-fatal error by the Application DMA.

When a non-fatal error occurs, the actions taken are detailed below:

- The ADMA prepare the Transfer Status field of the ABCR by setting the Transfer Complete bit and the appropriate error bit
- The ADMA writes the ABCR back to the descriptor pointed to by the ADAR.²²
- The ADMA begins processing the next chain descriptor(s), when any.

5.15 Power-Up/Default Status

Upon power-up, an external hardware reset, the Application DMA Registers are initialized to their default values.

22. When the Status Write Back Enable bit is set for a descriptor, the ABCR is written back to memory with the Transfer Complete bit set whether or not a non-fatal error has occurred.



5.16 Register Definitions

The Application DMA Unit contains seventy memory-mapped registers for controlling its operation. There is read/write access only to the ADMA Channel Control Register, the Internal Interface Parity Control Register, and the ADMA Next Descriptor Address Register. All other registers are read-only and are loaded with new values whenever an error condition occurs or from the chain descriptor whenever the ADMA reads a chain descriptor from memory.

The location of these registers are specified as a relative offset to a 512KB aligned global PMMR offset. The default for the 512KB aligned offset is 0_FFD8_0000H defined by the PMMRBAR register. See also [Chapter 21.0, "Peripheral Registers"](#). The Internal Bus Address of any ADMA register can be derived by adding the PMMRBAR plus the ADMA Address Offset and Registers offset.

For example, the internal bus address of the ADMA Channel Status Register for channel 2 (ADCSR2) would be 0_FFD8_0404H (0_FFD8_0000H + 0_0400H + 004H).

Table 314. ATU Internal Bus Memory Mapped Register Range Offsets

ADMA Unit	ADMA Address Offset (Relative to PMMRBAR)
ADMA0	+0 0000H
ADMA1	+0 0200H
ADMA2	+0 0400H

Table 315. Application DMA Unit Registers

Register Offset	Section, Register Name - Acronym (page)
+000H	Section 5.16.1, "ADMA Channel Control Register x — ACCRx" on page 521
+004H	Section 5.16.2, "ADMA Channel Status Register x — ACSRx" on page 522
+008H	Section 5.16.3, "ADMA Descriptor Address Register x — ADARx" on page 524
+018H	Section 5.16.4, "Internal Interface Parity Control Register x — IIPCRx" on page 524
+024H	Section 5.16.5, "ADMA Next Descriptor Address Register x — ANDARx" on page 525
+028H	Section 5.16.6, "ADMA Descriptor Control Register x — ADCRx" on page 526
+02CH	Section 5.16.7, "CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx" on page 529
+030H	Section 5.16.8, "ADMA Byte Count Register x — ABCRx" on page 530
+034H	Section 5.16.9, "Destination Lower Address / P_Destination Lower Address Register x — DLADRx" on page 531
+038H	Section 5.16.10, "Destination Upper Address / PQ_Destination Upper Address Register x — DUADRx" on page 532
+03CH through +0B8H	Section 5.16.11, "Source Lower Address Register 0...15_x — SLAR0...15_x" on page 533 Section 5.16.12, "Source Upper Address Register 0...15_x — SUAR0...15_x" on page 535

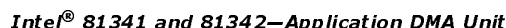


5.16.1 ADMA Channel Control Register x — ACCRx

The ADMA Control Register (ACCR) specifies parameters that dictate the overall operating environment. The ACCR should be initialized prior to all other ADMA registers following a system reset. Table 316 shows the register format. This register can be read or written while the ADMA is active.

Table 316. ADMA Channel Control Register x — ACCRx

Channel #	Internal bus address offset																													Attribute Legend:		RW = Read/Write	
0	0000H																													RV = Reserved		RC = Read Clear	
1	0200H																													PR = Preserved		RO = Read Only	
2	0400H																													RS = Read/Set		NA = Not Accessible	
Bit	Default	Description																															
31	0 ₂	Preserved																															
30	0 ₂	Preserved																															
29	INTERFACE_SEL_PCIX#	I/O Interface Select: This bit is read/write and used to select which ATU interface PCI-to-Memory operations use. 0 = When cleared, the ADMA uses the PCI-X interface for PCI-to-Local Memory transfers. 1 = When set, the ADMA uses the PCI Express interface for PCI-to-Local Memory transfers. Note: This bit automatically defaults to the interface selected by the INTERFACE_SEL_PCIX# strap. For single interface products, this means that the bit always remains in its' default state while for dual interface products, the user may choose to set this bit differently across the three ADMA channels.																															
28	0 ₂	Preserved																															
27:24	0H	Upper CRC Address -- This field represents bits[35:32] of the local memory address used by the ADMA for the CRC address defined for the CRC-32C operation. Note: This restricts the CRC-32C values referenced by the ADMA descriptors through the CRC Address to be located within the single 4 Gbyte address range defined by this register.																															
23:20	0H	Upper Next Descriptor Address -- This field represents bits[35:32] of the local memory address used to fetch the next descriptor. Note: This restricts the entire descriptor chain to be located within a single 4 Gbyte address range defined by this register.																															
19	0 ₂	Endian Mode Selector - This bit selects the Host vs. Local Memory Endian mode. This is used by the ADMA when the Endian Swap Enable in the ADCR is set. 0 = Host Memory is Little Endian, Local Memory is Big Endian 1 = Host Memory is Big Endian, Local Memory is Little Endian																															
18:02	00000H	Reserved																															
01	0 ₂	Chain Resume - when set, causes the ADMA to resume chaining by re-reading the current descriptor located at the address in the ADMA Descriptor Address Register when the ADMA is idle (ADMA Active bit in the ACSR is clear) or when the ADMA completes a transfer. This bit is cleared by hardware when either: - The ADMA completes a transfer and the ADMA Next Descriptor Address Register is non-zero. In this case, the ADMA proceeds to the next descriptor in the chain. - The ADMA re-reads the chain descriptor located at the address in the ADMA Descriptor Address Register and loads the Next Descriptor Address of that descriptor into the ADMA Next Descriptor Address Register																															
00	0 ₂	ADMA Enable - When set, the ADMA enables transfers. When clear, the ADMA disables any transfer. Clearing this bit when the ADMA is active suspends the current transfer at the earliest opportunity by halting all internal bus transactions. The ADMA does not initiate any new transfers when this bit is cleared. Data held in queues remains valid. Setting the bit after the ADMA is suspended causes the ADMA to resume the previously ongoing transfer.																															



The ADMA Channel Status Register (ACSR) contains status flags that indicate status. This register is typically read by software to examine the source of an interrupt. See [Section 5.14](#) for a description of the error conditions that are reported in the ACSR. See [Section 5.13](#) for a description of interrupts caused by the Application DMA.

Warning: Address out-of-range access made by the ADMA via the Direct Memory Port to the DDR Memory Controller is recorded as an **Address Region Error** by the DDR Memory Controller. The ADMA will continue processing the transfer as if no error occurred.

Bit	Default	Description
31:27	00000 ₂	Reserved.
26:14	000000H	Reserved
13	0 ₂	ADMA Active Flag - indicates the ADMA is either active (in use) or idle (available). When set, indicates the ADMA is in use and actively performing an operation. When clear, indicates the channel is idle and available to be configured for a new operation. The ADMA clears the ADMA Active flag when the previously configured transfer completes as a result of: • byte count reached zero and last chain descriptor is encountered (NULL value detected for Next Descriptor Address in chain descriptor) • Internal Bus Errors • When the Status Write Back Enable (bit 12 of the DC) is cleared, Zero Result Buffer Error, User Defined Tag Verification Error, or Data Guard Verification Error • Last chain descriptor is processed (NULL value detected for Next Descriptor Address in chain descriptor) and write enable is zero. The ADMA Active flag is set once a Chain Descriptor is read from memory.
12	0 ₂	End of Transfer Interrupt Flag - set when the ADMA has signalled an interrupt to the 81341 and 81342 after processing a descriptor but it is not the last descriptor in a chain.
11	0 ₂	End of Chain Interrupt Flag - set when the channel has signalled an interrupt to the 81341 and 81342 after processing a descriptor that is the last in a chain.
10	0 ₂	Reserved
09	0 ₂	Internal Interface Parity Error: Indicates that a data parity error was detected on the internal bus interface during a descriptor fetch. 0 = No error detected 1 = Error detected and recorded.
08:06	0 ₂	Reserved
05	0 ₂	MCU Port Abort Flag - This bit is set when during a read over the MCU port a Multi-bit ECC error is encountered.
04	0 ₂	Internal Bus Target Abort Flag - This bit is set on the occurrence of either a Host Master-Abort, a Host Target-Abort, Host Split Completion Error Message.



Table 317. ADMA Channel Status Register x — ACSRx (Sheet 2 of 2)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
Host Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na

Channel # Internal bus address offset Attribute Legend: RW = Read/Write
0 0004H RV = Reserved RC = Read Clear
1 0204H PR = Preserved RO = Read Only
2 0404H RS = Read/Set NA = Not Accessible

Bit	Default	Description
03	0 ₂	Internal Bus Master Abort Flag - This bit is set when the transaction initiated by the ADMA is not claimed on the internal bus, or due to an internal bus target device reporting an error - for example, when the target detects an address parity error on a request.
02	0 ₂	This bit is set when a Zero Result Buffer error is detected when Zero Result Buffer checking (or P+Q Zero Result Buffer Checking) is enabled in the ADMA and the Status Write Back Enable is cleared. For a P+Q Zero Result Buffer Check, this represents a non-zero result on the P Zero Result Buffer check.
01	0 ₂	This bit is set when a P+Q Zero Result Buffer error is detected on the Q Zero Result Buffer check when P+Q Zero Result Buffer checking is enabled in the ADMA and the Status Write Back Enable is cleared.
00	0 ₂	Reserved



5.16.3 ADMA Descriptor Address Register x — ADARx

The ADMA Descriptor Address Register (ADAR) contains the address of the current chain descriptor in local memory. This read-only register is loaded when a new basic or full chain descriptor is read. [Table 318](#) depicts the ADMA Descriptor Address Register. All chain descriptors are aligned on an 8 word (or 32 Byte) address boundary.

Table 318. ADMA Descriptor Address Register x — ADARx

		Diagram illustrating the bit fields of the ADARx register. The register is 32 bits wide, divided into two main sections: IOP Attributes (bits 31-4) and Host Attributes (bits 3-0). The IOP Attributes section contains 28 bits, all labeled 'ro' (Read Only). The Host Attributes section contains 4 bits, all labeled 'rv' (Read/Write). The bits are grouped into four sets of seven bits each, with bit numbers 31, 28, 24, 20, 16, 12, 8, and 4 indicated above the groups. The bit numbers 31, 28, 24, 20, 16, 12, 8, and 4 are also indicated below the groups. The bit numbers 31, 28, 24, 20, 16, 12, 8, and 4 are also indicated below the groups.																													
Channel #	Internal bus address offset															Attribute Legend:												RW = Read/Write			
0	0008 H															RV = Reserved												RC = Read Clear			
1	0208 H															PR = Preserved												RO = Read Only			
2	0408 H															RS = Read/Set												NA = Not Accessible			
Bit	Default	Description																													
31:05	0000 0000H	Current Descriptor Address - local memory address of the current chain descriptor read by the Application DMA.																													
04:00	00000 ₂	Reserved																													



5.16.5 ADMA Next Descriptor Address Register x — ANDARx

The ADMA Next Descriptor Address Register (ANDAR) contains the address of the next chain descriptor in local memory. When starting a transfer, this register contains the address of the first chain descriptor. Table 320 depicts the ADMA Next Descriptor Address Register.

All chain descriptors (Basic and Full) are aligned on an eight 32-bit word boundary. The ADMA may set bits 04:00 to zero when loading this register.

Note: The *ADMA Enable* bit in the ACCR and the *ADMA Active* bit in the ACSR must both be clear prior to writing the ANDAR. Writing a value to this register while the ADMA is active may result in undefined behavior.

Table 320. ADMA Next Descriptor Address Register x — ANDARx

Channel #	Internal bus address offset															Attribute Legend:																	
0	0024H															RW = Read/Write																	
1	0224H															RV = Reserved																	
2	0424H															PR = Preserved																	
																RS = Read/Set																	
																RC = Read Clear																	
																RO = Read Only																	
																NA = Not Accessible																	
Bit	Default	Description																															
31:05	0000000H	Next Descriptor Address - local memory address of the next chain descriptor to be read by the Application DMA.																															
04:00	00000 ₂	Reserved																															

5.16.6 ADMA Descriptor Control Register x — ADCRx

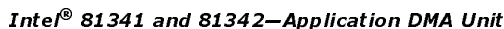
The ADMA Descriptor Control Register contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a basic or full chain descriptor is read from memory. These values may vary from chain descriptor to chain descriptor. The ADMA Table 321 shows the definition of the ADMA Descriptor Control Register.

Table 321. ADMA Descriptor Control Register x — ADCRx (Sheet 1 of 3)

Channel #	Internal bus address offset																														
0	0028H																														
1	0228H																														
2	0428H																														
		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																													
Bit	Default	Description																													
31	0 ₂	No Snoop Enable -- When set, the 81341 and 81342 sets the No Snoop (NS) attribute for requests initiated on the host I/O interface that are associated with this transfer.																													
30	0 ₂	Relaxed Ordering Enable -- When set, the 81341 and 81342 sets the Relaxed Ordering (RO) attribute for requests initiated on the host I/O interface that are associated with this transfer.																													
29:28	00 ₂	Reserved																													
27	0 ₂	Reserved																													
26:20	000000 ₂	Reserved																													
19	0 ₂	P Transfer Disable - When set and P+Q Transfer (or a P+Q Update Transfer or a P+Q Zero Result Buffer Error Check) is enabled (i.e., bit 18 (or bit 16 or bit 18 and bit 7) of the ADCRx is set), this bit disables the transfer of P to the P_Destination. Q is generated and transferred to the Q_Destination. Note: The hardware requires a properly aligned (16-byte address boundary) P_Destination Address be provided for proper operation even when the P transfer is disabled. When P transfer is disabled the hardware still performs the appropriate P operation, but simply does not write the result to the P_Destination Address.																													
18	0 ₂	P+Q Transfer Enable - When set, the ADMA generates both P and Q of the P+Q RAID6 algorithm on multiple source data streams (see Section 5.7.3, "XOR Operation with P+Q RAID-6" on page 494) and transfer them to the P_Destination and Q_Destination, respectively.																													
17	0 ₂	Dual XOR Enable - When set, the ADMA interprets this descriptor as a Dual XOR operation. The Source Selection Field (bits 6:3) must be programmed to 0011b when this enable bit is set. Note: The Dual XOR operation cannot be combined with any other ADMA operation.																													
16	0 ₂	P+Q Update Transfer Enable - When set, the ADMA interprets this descriptor as a P+Q Update transfer operation. The Source Selection Field (bits 6:3) must be programmed to 0011b when this enable bit is set. Note: The P+Q Update transfer operation cannot be combined with any other ADMA operation.																													
15	0 ₂	Reserved																													
14	0 ₂	Reserved																													
13	0 ₂	Endian Swap Enable - When set, the ADMA performs a 4 byte Endian Swap during a data transfers between the Host I/O interface and the Local Memory. The Endian assumptions for the Host and Local Memory are controlled by the Endian Mode Selector (bit 19 of the ACCR).																													
12	0 ₂	Status Write Back Enable - At the end of a transfer, when this bit is set, the ADMA writes back the Byte Count Word of the descriptor pointed to by the ADAR.																													
11	0 ₂	CRC Seed Fetch Disable -- When set and the CRC Generation Enable is set (bit 9 of the ADCR), the ADMA uses the byte-wise flipped and inverted version of the CRC value located at the CRC Address (CARMD) as the seed for the current CRC operation. When clear and the CRC Generation Enable is set (bit 9 of the ADCR), the ADMA uses the value located at the CRC Address (CARMD) as the seed for the current CRC operation.																													

**Table 321. ADMA Descriptor Control Register x — ADCRx (Sheet 2 of 3)**

		Channel #																Internal bus address offset																Attribute Legend:																RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RO = Read Only NA = Not Accessible															
		0																0028H																																															
		1																0228H																																															
		2																0428H																																															
Bit		Default		Description																																																													
10		0 ₂		CRC Transfer Disable -- When set and the CRC Generation Enable is set (bit 9 of the ADCR), the ADMA does not write the data stream to a destination while performing the CRC-32C generation. Note: When the Transfer Direction bits (ADCRx[02:01]) are setup as local memory destination and the CRC Transfer Disable bit is set (data stream not written to the destination), when a data parity error occurs on the internal bus during the transfer the error is not detected and logged. This is because data parity is forwarded by the ADMA to the Memory Controller where it is verified and logged in case a parity error is detected.																																																													
09		0 ₂		CRC Generation Enable -- When set, the DMA generates the CRC using the CRC-32C algorithm, the transferred data stream, and a seed value located at the CRC Address (CARMD). When the Data Transfer is complete, the ADMA writes the computed CRC value to CRC Address, and then write the BCR to the descriptor pointed to by the ADAR with the CRC Transfer Complete bit set (ABCR bit 31).																																																													
08		0 ₂		Memory Block Fill Enable - When this bit is set the ADMA fills memory with the 32-bit value contained in the Data Register (CARMD) based on the byte count in the ABCR register and the address in the XDAR register.																																																													
07		0 ₂		Zero Result Buffer Check Enable - When this bit is set, the ADMA computes the Zero Result Buffer check on the data blocks specified by the SUARx/SLARx registers. When the P+Q Transfer Enable is set, this bit enables a P+Q Zero Result Buffer Error Check.																																																													
06:03		0000 ₂		Source Selection - This field is used to select which SAR15:0 (SLAR15:0 / SUAR15:0 pair) registers are used by the XOR or Zero Result Buffer operations. 0000 SAR0 (1 Source) Data Transfer/CRC/Memory Block Fill 0001 SAR1:0 (2 Sources) XOR/Zero Result Check operation with 2 sources 0010 SAR2:0 (3 Sources) XOR/Zero Result Check operation with 3 sources 0011 SAR3:0 (4 Sources) XOR/Zero Result Check operation with 4 sources (Dual-XOR and P+Q Update) 0100 SAR4:0 (5 Sources) XOR/Zero Result Check operation with 5 sources 0101 SAR5:0 (6 Sources) XOR/Zero Result Check operation with 6 sources 0110 SAR6:0 (7 Sources) XOR/Zero Result Check operation with 7 sources 0111 SAR7:0 (8 Sources) XOR/Zero Result Check operation with 8 sources 1000 SAR8:0 (9 Sources) XOR/Zero Result Check operation with 9 sources 1001 SAR9:0 (10 Sources) XOR/Zero Result Check operation with 10 sources 1010 SAR10:0 (11 Sources) XOR/Zero Result Check operation with 11 sources 1011 SAR11:0 (12 Sources) XOR/Zero Result Check operation with 12 sources 1100 SAR12:0 (13 Sources) XOR/Zero Result Check operation with 13 sources 1101 SAR13:0 (14 Sources) XOR/Zero Result Check operation with 14 sources 1110 SAR14:0 (15 Sources) XOR/Zero Result Check operation with 15 sources 1111 SAR15:0 (16 Sources) XOR/Zero Result Check operation with 16 sources When using the Dual-XOR operation or P+Q Update operation, this field must be programmed to 0011b. Note: When more than one source is chosen and the Zero Result Buffer Check Enable is cleared, the ADMA performs an XOR operation.																																																													



IOP Attributes	31	28		24		20		16		12		8		4		0
	ro	ro	rv	rv	rv	rv	rv	rv	rv	rv	ro	ro	ro	ro	ro	ro
Host Attributes																
	na	na	na	na	na	na	na	na	na	na	na	na	na	na	na	na

Channel # Internal bus address offset

0 0028 H

1 0228 H

2 0428 H

Attribute Legend:

RW = Read/Write

RV = Reserved

PR = Preserved

RO = Read Only

RS = Read/Set

NA = Not Accessible

Bit	Default	Description										
02:01	00 ₂	Transfer Direction - This field is used to selects the ADMA transfer type. <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%; text-align: left;"><u>Source</u></th> <th style="width: 50%; text-align: left;"><u>Destination</u></th> </tr> </thead> <tbody> <tr> <td>00 Host I/O interface</td> <td>Local Memory</td> </tr> <tr> <td>01 Local Memory</td> <td>Host I/O interface</td> </tr> <tr> <td>10 Internal Bus</td> <td>Local Memory</td> </tr> <tr> <td>11 Local Memory</td> <td>Internal Bus</td> </tr> </tbody> </table> Note: When the Source Selection field does not equal 0000b (Single Source), only the Local Memory may be selected as the Source (Transfer Direction = 01b, or Transfer Direction = 11b) Note: Selection of a Transfer Direction using the Host I/O interface directly maps the programmed range of addresses (Source/Destination) over to the Host I/O interface including the upper 32-bits of the Host I/O interface address programmed in the descriptor. Note: When a Transfer Direction using the Internal Bus or Local Memory is selected, the programmed range of addresses (Source/Destination) is direct mapped from the ADMA onto the Internal Bus or Local Memory including the upper 4-bits of the address programmed in the descriptor.	<u>Source</u>	<u>Destination</u>	00 Host I/O interface	Local Memory	01 Local Memory	Host I/O interface	10 Internal Bus	Local Memory	11 Local Memory	Internal Bus
<u>Source</u>	<u>Destination</u>											
00 Host I/O interface	Local Memory											
01 Local Memory	Host I/O interface											
10 Internal Bus	Local Memory											
11 Local Memory	Internal Bus											
00	0 ₂	Interrupt Enable - When set, the Application DMA generates an interrupt to the 81341 and 81342 upon completion of a transfer. When clear, no interrupt is generated.										



5.16.7 CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx

When CRC generation is enabled in the ADMA (bit 10 of ADCR), the CRC Address Register (CARx) refers to the location where the CRC Engine loads the seed value and write out the CRC value following the completion of the Data Transfer. This register is read-only. [Table 322](#) depicts the CRC Address Register. The upper 4-bit of the 36-bit address is located in the ADMA Channel Control Register. These 4-bits must be programmed in the ACCR prior to initiating the descriptor chain.

When a Memory Block Fill operation is enabled in the ADMA, this register is used as the 32-bit constant used to fill memory.

When a P+Q Transfer is enabled in the ADMA, this register is used as the lower 32-bit address of the Q_Destination.

All chain descriptors are aligned on an eight 32-bit word boundary.

Table 322. CRC Address/Memory Block Fill Data/Q_Destination Register x – CARMDQx

Channel #	Intel XScale® processor internal bus address offset	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set
0	002CH	RC = Read Clear
1	022CH	RO = Read Only
2	042CH	NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	CRC Address Register - When CRC Mode is enabled, this address is used to load the seed value prior to the Data Transfer and write out the CRC value following the completion of the Data Transfer. Note: When operating in CRC mode, the CRC Address is a 32-bit aligned value, and bits 1:0 are reserved. For a Memory Block Fill operation, this 32-bit value is used to fill memory. When a P+Q Transfer is enabled, this 32-bit value is used as the lower 32-bit address of the Q_Destination



5.16.8 ADMA Byte Count Register x — ABCRx

The ADMA Byte Count Register (ABCR) contains the number of bytes to transfer for an operation and an operation status field which are updated when the Status Write Back Enable in the ADCR is set. This is a read-only register that is loaded from the Byte Count word in a basic or full chain descriptor. It allows for a maximum transfer of 16 Mbytes. A value of zero is a valid byte count and results in no data transfers.

Note: Anytime this register is read by the 81341 and 81342, it contains the number of bytes left to transfer. Note that during an operation valid data may be present in the Application DMA store queue. This register is decremented by 1 Kbyte granularity when all of the programmed operations have completed on 1 Kbytes of all enabled sources, or simply decremented to 0000H when the byte count is less than 1 Kbytes.

Table 323. ADMA Byte Count Register x — ABCRx

Channel #	Internal bus address offset																			Attribute Legend:		RW = Read/Write		RC = Read Clear		PR = Preserved		RO = Read Only		RS = Read/Set		NA = Not Accessible					
0	0030H																																				
1	0230H																																				
2	0430H																																				

Bit	Default	Description
31	0 ₂	Transfer Complete - This bit is set when the ADMA completes the processing of the descriptor. Note: The ADMA updates this status in memory only by updating the Byte Count Word of the current descriptor (the fourth word of the descriptor pointed to by the ADAR).
30	0 ₂	Zero Result Buffer Error - This bit is set when the bit wise XOR computed across the data blocks specified by the SARx registers results in the detection of a non-zero result (i.e., an invalid parity block). For a P+Q Zero Result Buffer Check, this represents an error on the P check data. Note: The ADMA updates this status in memory only by updating the Byte Count Word of the current descriptor (the fourth word of the descriptor pointed to by the ADAR).
29	0 ₂	Zero Result Buffer Error Q - For a P+Q Zero Result Buffer check, this bit is set when the bit wise XOR computed across the Galois multiplied data blocks specified by the SARx registers results in the detection of a non-zero result (i.e., an invalid Q block). Note: The ADMA updates this status in memory only by updating the Byte Count Word of the current descriptor (the fourth word of the descriptor pointed to by the ADAR).
28	0 ₂	Reserved
27	0 ₂	Reserved
26:24	000 ₂	Host I/O Interface Function Number - This field is used to program the host I/O interface function number (0-7) that the current descriptor is associated with.
23:00	000000H	Byte Count - is the number of bytes to transfer for an operation.



5.16.9 Destination Lower Address / P_Destination Lower Address Register x — DLADRx

For data transfer operations, the Destination Lower Address Register (DLADRx) contains the lower 32-bits of a 64-bit destination address. [Table 324](#) shows the register. During XOR-transfer operations, this address is the destination address where the XOR data stream is stored.

For Dual XOR operations, this register contains the lower 32-bits of the 64-bit Horizontal Destination Address.

For P+Q Update operations, this register contains the lower 32-bits of the 64-bit P Destination Address.

During Memory Block Fill operations, this address points to the memory block to be written with the constant value contained in the "CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx" on [page 529](#).

For a P+Q Transfer, this register is used for the lower 32-bits of the P_Destination.

This register is read-only and is loaded when a basic or full chain descriptor is read from memory.

Table 324. Destination Lower Address Register / P_Destination Lower Address x — DLADRx

		Intel XScale® processor internal bus address																Attribute Legend:															
Channel #		offset																RW = Read/Write															
0		0034H																RV = Reserved															
1		0234H																RC = Read Clear															
2		0434H																PR = Preserved															
																		RO = Read Only															
																		RS = Read/Set															
																		NA = Not Accessible															
Bit	Default	Description																															
31:00	00000000H	Destination Lower Address - This 32-bit value is the Destination Lower address, for an XOR operation, Memory Block Fill operation, and the P_Destination lower 32-bit address for a P+Q operation.																															



5.16.10 Destination Upper Address / PQ_Destination Upper Address Register x — DUADRx

For data transfer operations, the Destination Upper Address Register (DUADRx) contains the upper 32-bits of a 64-bit destination address. Table 325 shows the register. During XOR-transfer operations, this address is the destination address where the XOR data stream is stored.

For Dual XOR operations, this register contains the upper 32-bits of the 64-bit Horizontal Destination Address.

For P+Q Update operations, this register contains the upper 32-bits of the 64-bit P Destination Address.

During Memory Block Fill operations, this address points to the memory block to be written with the constant value contained in the “CRC Address/Memory Block Fill Data/Q_Destination Register x — CARMDQx” on page 529.

For a P+Q Transfer, this address contains the upper 32-bits of both the P_Destination and the Q_Destination.

This register is read-only and is loaded when a basic, full or Dual XOR chain descriptor is read from memory.

Note: When the Transfer Direction field (bits 2:1 of the ADCR) maps the destination address to the Host I/O interface, this register represents bits 63:32 of the Host address. Otherwise, bits 3:0 of this register represents bits 35:32 of the internal bus or local memory address.

Warning: When the Transfer Direction field (bits 2:1 of the ADCR) maps the destination address to the internal bus or local memory interface address, bits 31:4 of the DUADRx **must** be programmed to all zeros.

Table 325. Destination Upper Address / PQ_Destination Upper Address Register x — DUADRx

		312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Host Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na																															
Channel #		Intel XScale® processor internal bus address offset																Attribute Legend:								RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible							
0		0038 H																															
1		0238 H																															
2		0438 H																															
Bit	Default	Description																															
31:00	00000000H	Destination Upper Address - This 32-bit value is the upper 32-bit destination address. For a P+Q Transfer, this 32-bit value represents the upper 32-bit address for both the P_Destination and the Q_Destination.																															



5.16.11 Source Lower Address Register 0...15_x — SLAR0...15_x

The 81341 and 81342 Source Lower Address Register0...15 (SLAR0...15) contains the lower 32-bit source address. There are 16 Source Address Registers (SLAR0 - SLAR15). Each of these registers is loaded with the address of blocks of data to be operated upon by the ADMA. The ADCR register controls the operation performed on the data blocks referenced by the registers (SLAR0 - SLAR15).

For Dual XOR Operations, the following SLAR0...15_x registers have been redefined:

SLAR0_x	Lower 32-bit address of the first source data block
SLAR1_x	Lower 32-bit address of the second source data block
SLAR2_x	Lower 32-bit address of the Horizontal source data block
SLAR3_x	Lower 32-bit address of the Diagonal source data block
SLAR4_x	Lower 32-bit address of the Diagonal destination block

For P+Q Update Operations, the following SLAR0...15_x registers have been redefined:

SLAR0_x	Lower 32-bit address of the first source data block
SLAR1_x	Lower 32-bit address of the second source data block
SLAR2_x	Lower 32-bit address of the P source data block
SLAR3_x	Lower 32-bit address of the Q source data block
SLAR4_x	Lower 32-bit address of the Q destination block

Reading SLARx registers once ADMA has started a chain descriptor returns the current source addresses. Once an operation is initiated, these registers contain current source addresses. For example; when Byte Count is initially 4096 bytes and ADMA has completed operation on the first three 1K-byte data blocks, the value in register SARx is the equal to the programmed descriptor value + 3072 (SARx + 3072).

Table 326 shows the Source Lower Address Register0...15. These read-only registers are loaded when a basic, full, P+Q, or Dual XOR chain descriptor is read from memory.

Note: This register represents bits 31:00 of the local memory address.

Note: The register addresses for the Sources Lower Address Registers (SLARx) and the Source Upper Address Registers (SUARx) are ordered the same as in the descriptor formats.

Table 326. Source Lower Address Register 0...15_x — SLAR0...15_x

Register	Ch-0	Ch-1	Ch-2
SLAR0	003CH	023CH	043CH
SLAR1	0044H	0244H	0444H
SLAR2	004CH	024CH	044CH
SLAR3	0054H	0254H	0454H
SLAR4	005CH	025CH	045CH
SLAR5	0064H	0264H	0464H
SLAR6	006CH	026CH	046CH
SLAR7	0074H	0274H	0474H
SLAR8	007CH	027CH	047CH
SLAR9	0084H	0284H	0484H
SLAR10	008CH	028CH	048CH
SLAR11	0094H	0294H	0494H
SLAR12	009CH	029CH	049CH
SLAR13	00A4H	02A4H	04A4H
SLAR14	00ACH	02ACH	04ACH
SLAR15	00B4H	02B4H	04B4H



5.16.12 Source Upper Address Register 0...15_x — SUAR0...15_x

The 81341 and 81342 Source Upper Address Register0...15 (SUAR0...15) contains the Upper 32-bit source address. There are 16 Source Address Registers (SUAR0 - SUAR15). Each of these registers is loaded with the address of blocks of data to be operated upon by the ADMA. The ADCR register controls the operation performed on the data blocks referenced by the registers (SUAR0 - SUAR15).

For Dual XOR Operations, the following SUAR0...15_x registers have been redefined:

SUAR0_x	Upper 32-bit address of the first source data block
SUAR1_x	Upper 32-bit address of the second source data block
SUAR2_x	Upper 32-bit address of the Horizontal source data block
SUAR3_x	Upper 32-bit address of the Diagonal source data block
SUAR4_x	Upper 32-bit address of the Diagonal destination block

For P+Q Update Operations, the following SUAR0...15_x registers have been redefined:

SUAR0_x	Upper 32-bit address of the first source data block
SUAR1_x	Upper 32-bit address of the second source data block
SUAR2_x	Upper 32-bit address of the P source data block
SUAR3_x	Upper 4-bit address of the Q source data block and the 8-bit Q Update Multiplier (DMLTQ)
SUAR4_x	Upper 32-bit address of the Q destination block



Table 327 shows the Source Upper Address Register0...15. These read-only registers are loaded when basic, full, P+Q, or Dual XOR chain descriptor is read from memory.

For a P+Q Transfer, the upper 8-bits of these registers represent the 16 Galois Field Multipliers (see Section 5.7.3, “XOR Operation with P+Q RAID-6” on page 494) associated with the 16 Source Addresses.

Note: When the Transfer Direction field (bits 2:1 of the ADCR) maps the source address(es) to the Host I/O interface, this register represents bits 63:32 of the Host address. Otherwise, bits 3:0 of this register represents bits 35:32 of the internal bus or local memory address.

Warning: When the Transfer Direction field (bits 2:1 of the ADCR) maps the source address(es) to the internal bus or local memory interface, bits 31:4 of the SUAR0...15_x **must** be programmed to all zeros.

Note: The register addresses for the Sources Upper Address Registers (SUARx) and the Source Upper Address Registers (SUARx) are ordered the same as in the descriptor formats.

Table 327. Source Upper Address Register 0...15_x — SUAR0...15_x

				312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Host Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na																											
				Internal Bus Address Offset																											
Register				Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																											
	Ch-0	Ch-1	Ch-2																												
SUAR0	0040H	0240H	0440H																												
SUAR1	0048H	0248H	0448H																												
SUAR2	0050H	0250H	0450H																												
SUAR3	0058H	0258H	0458H																												
SUAR4	0060H	0260H	0460H																												
SUAR5	0068H	0268H	0468H																												
SUAR6	0070H	0270H	0470H																												
SUAR7	0078H	0278H	0478H																												
SUAR8	0080H	0280H	0480H																												
SUAR9	0088H	0288H	0488H																												
SUAR10	0090H	0290H	0490H																												
SUAR11	0098H	0298H	0498H																												
SUAR12	00A0H	02A0H	04A0H																												
SUAR13	00A8H	02A8H	04A8H																												
SUAR14	00B0H	02B0H	04B0H																												
SUAR15	00B8H	02B8H	04B8H																												

Bit	Default	Description
31:24	00H	Data Multiplier (DMLT0...15) - In a P+Q operation, this value is the 8-bit Galois field multiplier associated with Source 0...15, otherwise these bits are Reserved. In a P+Q update operation, this is the 8-bit Galois field multiplier associated with the Q check data block update function. This only applies to SUAR3 in this operation.
23:00	00000000H	Upper Memory Address - Upper 4-bit Local memory source address.



6.0 System Controller (SC) and Internal Bus Bridge

This chapter describes the System Controllers (SC) of the Intel® 81341 and 81342 I/O Processors (81341 and 81342). The System Controller controls the internal bus and its agents. There are two System Controllers on 81341 and 81342 since there are two internal busses.

6.1 Overview

The System Controller controls the internal bus agents arbitrating for the internal bus. The Internal Bus on 81341 and 81342 contains a separate address bus arbiter and data bus arbiter as the address and data busses are completely de-multiplexed. There are two internal busses on 81341 and 81342 and therefore there are two system controllers implemented — one for the North Internal Bus and one for the South Internal Bus.

- The north internal bus SC controls the two Intel XScale® processors, the DDR Memory Controller, the Bridge, and the SAS Interface.
- The south internal bus SC controls the ATU-E, ATU-X, the Bridge, the DDR SDRAM Memory Controller, the Application DMAs, the PBI, and the APB interface.

In addition to providing the address bus and data bus arbitration functionality, the SC also initiates address and data transactions once the bus has been granted. The SC is also the central hub which takes as inputs all the agents address and data busses, and then controlling how the address bus or data bus is routed to an agent.

The SC also provides hardware functionality that can be used to force address and data parity errors. This feature allows software to test error handling routines by forcing address or data parity error.



6.2 Theory of Operation

6.2.1 System Controller

The XSI System Controller (SC) is the arbiter for the XSI bus. There is one SC for the North XSI bus and one for the South XSI bus since there are two XSI busses on 81341 and 81342. The XSI bus supports fully demultiplex and independent address and data paths, thus the SC performs arbitration for address requests and data requests separately. Up to 15 agents are supported by the SC.

For address request arbitration, the SC receives address bus request from each requesting agent seeking to master requests on the address bus and implements an arbitration algorithm to generate address bus grant to grant the bus to a particular requesting agent. The SC actually frames the address transactions, and therefore drives the address strobe along with the address bus grant. The assertion of address strobe indicates that a valid XSI bus command is driven the following cycle.

For data request arbitration, the SC receives data bus request from each requesting agent seeking to master a data transaction on the data bus and implements an arbitration algorithm to generate data bus grant to grant the data bus to a particular requesting agent. The SC frames the data transactions, and therefore initiates the data transaction along with the data bus grant.

The SC ORs several bus outputs from each agent and routes the ORed signals back to each agent, including the Address Qualifier Signals (AQS) and Data Qualifier Signals (DQS).

Both the address and data bus arbiters support simple round-robin algorithms. Each internal bus initiator can be individually enabled or disabled. When an initiator is disabled, any address request made by that initiator is not granted the internal bus. Each arbiter parks on the last master.

6.2.2 Internal Bus Requester IDs

Each of the initiator/requester on the 81341 and 81342 has an assigned unique ID, which helps identify an initiator when returning read data and for the purpose of logging transaction errors. [Table 328](#) lists the encoded initiator IDs.

Table 328. Intel® 81341 and 81342 I/O Processors Initiator IDs

Internal Bus Initiator	Initiator ID
Reserved	0000 ₂
Intel XScale® processor 0 (coreID = 0)	0001 ₂
Intel XScale® processor 1 (coreID = 1)	0010 ₂
ATU-X	0011 ₂
ATU-E	0100 ₂
ADMA	0101 ₂
Reserved	0110 ₂
Messaging Unit ^a	0111 ₂
Reserved	1000 ₂
SMBus	1001 ₂
Reserved	1010 ₂
Reserved	1011 ₂ through 1111 ₂

a. The Messaging Unit acts as internal bus initiator only for issuing MSI or MSI-X writes to the ATUE or ATUX.



6.2.3 Parity Testing

The 81341 and 81342 supports parity protections on both the 36-bit address and 128-bit data bus south internal bus. Parity is supported on a byte-wise basis. The SC provides hardware test features that allows the user to force address or data parity errors on the internal bus. This feature allows the user to test software error handling routines by forcing an address or data parity error on the internal bus. Parity error is forced by XORing (inverting) good parity bit or bits before they are driven on the bus. The 81341 and 81342 provides separate sets of error registers for injecting address bus and data bus parity error. The 81341 and 81342 provides an address mask register and a data mask register. The user can program the mask registers to select which parity bit(s) ought to be inverted. 81341 and 81342 also provides a register that allows the programmer to select the initiator of a given transaction by providing the ID of the initiator.

This register also provides an enable bit which must be set by software and is reset by hardware. This provides a way of injecting an error only once — a one-shot process. For example, error is injected only during the address cycle for an address parity test, and during the first data cycle for a data parity test. [Table 330](#) lists the Initiator IDs that must be programmed in the *Initiator ID* field of SIBATCR for address parity testing during address request. [Table 330](#) lists the Initiator IDs that must be programmed in the *Initiator ID* field of SIBDTCR for data parity testing during writes. [Table 330](#) lists the Initiator IDs that must be programmed in the *Initiator ID* field of SIBDTCR for data parity testing during read completions.

The initiator in this context is used to identify the source of the address or data. For example, when data parity error is to be injected while the DDR MCU is returning read completion data, the DDR MCU Initiator ID must be used. However, when data parity error is to be injected when the ATU is writing data to the DDR Memory, the ATU Initiator ID must be used.

Figure 76. Typical Internal Bus System Controller Block Diagram

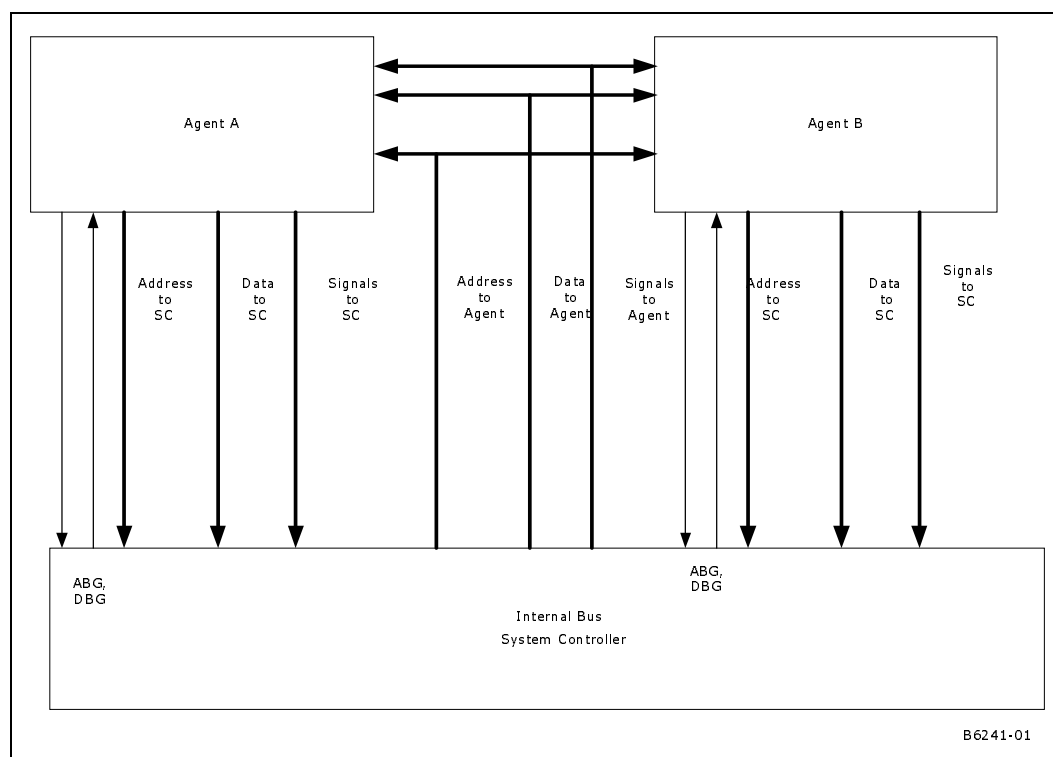




Table 329. Address and Data Parity Testing Initiator IDs

Internal Bus Initiator	Initiator ID
Reserved	0000 ₂
Intel XScale™ processor 0 (coreID = 0)	0001 ₂
Intel XScale™ processor 1 (coreID = 1)	0010 ₂
ATU-X	0011 ₂
ATU-E	0100 ₂
ADMA	0101 ₂
Reserved	0110 ₂
Messaging Unit	0111 ₂
Reserved	1000 ₂
SMBus and PMON	1001 ₂
Reserved	1010 ₂
Reserved	1011 ₂ through 1111 ₂

Note: This table contains the Initiator IDs for injecting address parity error when these initiators are making address requests. In addition these same Initiator IDs can be used when injecting data parity error when these initiators are pushing data during writes.

Table 330. Data Parity Testing Completer IDs

Internal Bus Initiator	Initiator ID
Reserved	0000 ₂
Not Applicable ^a	0001 ₂
Not Applicable	0010 ₂
ATU-X	0011 ₂
ATU-E	0100 ₂
ADMA	0101 ₂
Reserved	0110 ₂
Messaging Unit and DDR MCU	0111 ₂
Reserved	1000 ₂
UART, I2C, GPIO, PBI, and PMON	1001 ₂
Not Applicable	1010 ₂
Reserved	1011 ₂ through 1111 ₂

Note: This table contains the Initiator IDs for injecting data parity error when these Initiators are returning data during read completions. Note that in this scenario the initiator of the data transaction is the actual completer.

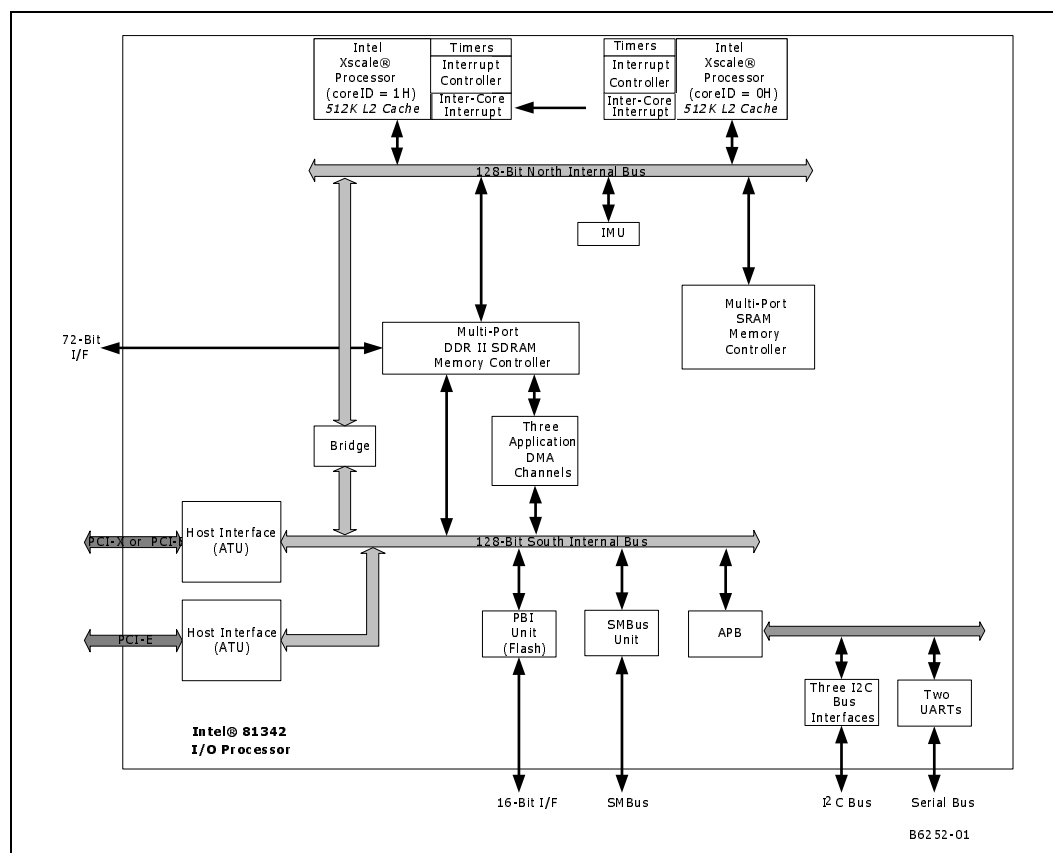
- a. Not applicable implies that the ID associated with that initiator does not return completion data.



6.3 Internal Bus Bridge

This section describes the internal bus bridge. The internal bus bridge isolates traffic on the north internal bus and the south internal bus. The internal bus bridge is a bidirectional bridge. Transactions targeting the south internal bus from the north internal bus are referred to as **outbound transactions**. Transactions targeting the north internal bus from the south internal bus are referred to as **inbound transactions**.

Figure 77. Intel® 81341 and 81342 I/O Processors Block Diagram (External)





6.3.1 Theory of Operation

The bridge forwards north internal bus address requests that are not claimed on the north internal bus. For example, the bridge performs subtractive decoding on the north internal bus interface. The bridge claims and forwards a north internal bus data transaction when the ID provided with the data transaction matches the ID of a previously claimed write address request.

On the south internal bus, the XBG defines a Bridge Memory Window. Transactions on the south internal bus that target the Bridge Memory Window are claimed and forwarded to the north internal bus. The bridge claims and forwards a south internal bus data transaction when the ID provided with the data transaction matches the ID of a previously claimed write address request. The Bridge south interface also claims transactions that target the Bridge memory-mapped registers. Intel XScale® processor transactions that target the Bridge memory-mapped registers are also propagated to the south internal bus, and then claimed by the Bridge south interface.

Both the north and south internal busses on 81341 and 81342 support the same bus protocol. The internal bus operates by performing split transactions on both read and write address requests. Every address request contains an Address Transfer ID (ATID), which is the ID of the initiator. Every initiator has a unique ATID. When an initiator makes a read request, it drives its ATID along with the address request. Once the Bridge claims the request, it maintains the ATID, which it uses when returning read completion data in the form of a Data Transfer ID (DTID). The initiator of the read request claims the read completion transaction by observing that the DTID matches its ATID. For a write request, the initiator drives its ATID along with the address request. Once the Bridge claims the request, it maintains the ATID, which it uses to claim a data transaction. The initiator of the write request drives a Data Transfer ID (DTID) with the data transaction. The Bridge claims the data transaction by observing that the DTID matches the stored ATID.

Each of the initiator/requester on the 81341 and 81342 has an assigned unique ID, which helps identify an initiator when returning read data and for the purpose of logging transaction errors. [Table 328](#) lists the encoded initiator IDs. The bridge uses the same initiator ID of a transaction it claims when forwarding the transaction on the opposite internal bus.



6.3.2 Internal Bus Commands

Table 331 lists the internal bus commands that are supported on the north and south bridge interfaces.

Table 331. Bridge supported Internal Bus Commands

Internal Bus Command Encoding	Internal Bus Command Type	Claimed on North Internal Bus	Generated on South Internal Bus	Claimed on South Internal Bus	Generated on North Internal Bus
0000	NULL	No	Yes	No	Yes
0001	Sync	No	No	No	No
0010	Special	No	No	No	No
0011	Reserved	No	No	No	No
0100	Reserved	No	No	No	No
0101	Reserved	No	No	No	No
0110	Reserved	No	No	No	No
0111	Reserved for TLBIE	No	No	No	No
1000	Read	Yes	Yes	Yes	Yes
1001	Read Line	Yes	Yes	Yes	Yes
1010	Reserved for Invalidate Line	No	No	No	No
1011	Read and Invalidate Line	Yes	Yes	Yes	Yes
1100	Write	Yes	Yes	Yes	Yes
1101	Reserved for Clean and Invalidate Line	No	No	No	No
1110	Write Line	Yes	Yes	Yes	Yes
1111	Reserved for Clean Line	No	No	No	No

6.3.3 Transaction Queues

Both the north and south interfaces of the bridge support a read queue of 8 entries and each supporting up to 32 Bytes of data buffers. Both interfaces support a write queue of 8 entries, each supporting up to 32 Bytes of data buffers.

Note: The bridge master-aborts any transaction request that tries to cross a 32-byte boundary. Since each data buffer is 32 bytes in size, the bridge can only transfer a maximum byte-count of 32 bytes of data per request and only when the address is aligned on a 32-byte boundary. In other words, for a non 32-byte aligned address, the sum of the non-aligned address and byte-count has to be less than the next 32-byte aligned address boundary for the bridge to enqueue the request.



6.3.4 Bridge Memory Window

The North Internal Bus interface of the Bridge performs subtractive decoding. For example, transactions on the north internal bus that are not claimed by other targets on the north internal bus are claimed by the Bridge North Interface.

The South Bridge Interface performs positive decoding. The Bridge provides a Bridge Memory Window defined by the [Bridge Window Base Address Register — BWBAR](#) and the [Bridge Limit Register — BWLR](#). Transactions on the south internal bus that target this memory window are claimed and forwarded to the north internal bus. The South Bridge Interface also claims transactions that target the Bridge memory-mapped registers. The memory-mapped registers are only accessible from the south internal bus. For example, transactions to the Bridge memory-mapped registers by the Intel XScale® processors that reside on the north internal bus, are propagated from the north internal bus to the south internal bus via the Bridge, and then claimed by the Bridge on the south interface.

Note: For bridge memory window access overlapping the bridge memory-mapped register space (+1780 through +17FFH), the bridge memory window is not accessible, and the bridge memory-mapped register space is addressed.



6.3.5 Ordering and Passing Rules

Table 332 lists the ordering and passing rules requirements for the bridge. Although write requests and write data completions are completely independent transactions on the internal bus, the bridge internally combines a write request with its corresponding write data transaction when enqueueing write requests and when issuing write requests. For example, a write request made to the target side of the bridge is not considered valid in the bridge until the write request and the data for that write request have been received. Similarly, a write request mastered by the bridge is not considered done until the write request and the write data completion have been executed on the internal bus. For that reason, Table 332 shows a write request (Addr(Wr)) and a write data completion (Data(Wr)) as one entity.

- Addr(Wr) — Write Address Request
- Addr(Rd) — Read Address Request
- Data(Rd) — Read Data Completion
- Data(Wr) — Write Data Completion

Table 332. Ordering and Passing Rules for both Inbound and Outbound Transactions

Row Pass Column?	Addr(Wr)/Data(Wr) (Column 1)	Addr(Rd) (Column 2)	Data(Rd) (Column 3)
Addr(Wr)/Data(Wr) (Row 1)	No (Row 1, Column 1)	Yes ^a (Row 1, Column 2)	Yes/No (Row 1, Column 3)
Addr(Rd) (Row 2)	No (Row 2, Column 1)	No (Row 2, Column 2)	Yes/No (Row 2, Column 3)
Data(Rd) (Row 3)	No (Row 3, Column 1)	Yes (Row 3, Column 2)	Yes/No (Row 3, Column 3)
No - The row is <i>not</i> allowed to pass the column Yes - The row must be allowed to pass the column Yes/No - There are no requirements			

a. Strong Ordering Rule Requirements for Outbound Write Requests in order to maintain data coherency. Refer to Section 6.3.5.1, Strong Ordering Rule Requirements.

6.3.5.1 Strong Ordering Rule Requirements

This rule applies for Write Requests targeting only the north interface of the Bridge. For example, outbound write requests from the north internal bus to the south internal bus. Although the bridge must allow write requests to pass read requests as shown in Table 332 (Row 1, Column 2), to maintain data coherency the bridge does not allow enqueueing a write request whose cacheline address matches that of a previously enqueued read request. The write request is retried on the north internal bus until the read request has been retired. For example, the data for the read request has been returned.

Note: The south interface of the South Bridge does not follow the Strong Ordering Rule Requirements described in Section 6.3.5.1, Strong Ordering Rule Requirements.



6.3.6 Parity Support

The bridge supports parity as required by the south internal bus. The south internal bus supports both byte-wise address and data parity. Therefore, as a initiator the bridge is responsible to drive byte-wise parity on the south internal bus on both the 36-bit address bus and the 128-bit data bus. Also when completing read requests for south internal bus initiators, the bridge drives data parity. The south internal bus supports even parity. Note that the north internal bus does not support any parity. The bridge also supports byte-wise parity on the internal data buffers.

6.3.6.1 Address Parity Generation

Only the bridge south interface generates byte-wise address parity on address it initiates on the south internal bus.

6.3.6.2 Address Parity Checking

Only the south interface of the bridge verifies address parity when claiming south internal bus write transactions.

6.3.6.3 Data Parity on Outbound Transactions

For an outbound transaction (transaction flowing from the north internal bus to the south internal bus as either a read completion or write request), the bridge generates data parity as the data enters the north bridge interface. The data and its parity are stored in the internal data buffers. When the transaction is initiated on the south internal bus, the bridge simply drives the data along with the parity as stored in the data buffers. For example, the bridge does not generate parity. The receiver of the data on the south internal bus verifies the data parity.

6.3.6.4 Data Parity on Inbound Transactions

For an inbound transaction (transaction flowing from the south internal bus to the north internal bus as either a read completion or a write request), the bridge simply writes the data along with the received data parity to the internal data buffers. The bridge then checks the data parity while forwarding the transaction to the north internal bus. When the bridge detects a parity error on a write transaction, the bridge logs the error and also forwards the transaction on the north internal bus. When the bridge detects a parity error on a read completion, the bridge logs the error and assert DABORT instead of completing the transaction on the north internal bus. Refer to the following error logging registers: "[Bridge Error Control and Status Register — BECSR](#)", "[Bridge Error Address Register — BERAR](#)" and the "[Bridge Error Upper Address Register — BERUAR](#)"



6.3.7 Error Detection and Handling

The Bridge provides a set of error logging registers that are used to log any error that are encountered by the Bridge: north interface or south interface. Only one error is logged, when more errors occur when one error is already logged, the bridge would indicate that it detected more errors, but does not log these newer errors.

The bridge drives a single interrupt (South Internal Bus Bridge Error Interrupt Pending) to the Interrupt Controller Unit (ICU). An interrupt may be generated by the bridge to report any error condition detected to the Intel XScale® processor by setting bit 16 of BECSR. Whenever the bridge toggles bit 0 of BECSR from 0 to 1, an interrupt is generated to the core.

6.3.7.1 Bridge North Internal Bus Interface Error

The following conditions may be encountered by the Bridge North Interface:

- Master Abort on the North Internal Bus interface. This condition may happen when the bridge attempts an address request on the north internal bus and the request is not claimed by any target. The bridge would log the error as a Master Abort.
 - For a write request, the bridge simply logs the master abort error condition and discard the write data.
 - For a read request, the Bridge logs the Master Abort. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the south internal bus that it encountered an error by performing a target abort during the read completion.
- Address Request Error on the north internal bus interface. This condition may happen when a target on the north internal bus detected an error during the address request. For example, the target may indicate that the byte-count is out of range.
 - For a write request, the bridge logs the address request error condition and discard the write data.
 - For read request, the Bridge logs the address request error. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the south internal bus that it encountered an error by performing a target abort during the read completion.
- Target Abort on the north internal bus. This condition may occur when a target claimed a previously issued read request, but is unable to return the read completion data. The bridge would log the error as a target abort.
 - The Bridge logs the target abort. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the south internal bus that it encountered an error by performing a target abort during the read completion. Target Abort on the north internal bus. This condition may occur when the north interface detects a parity error during the data phase of a write request to the north internal bus, or during a read completion to the north internal bus.
 - For a read completion, the Bridge logs the target abort condition and completes the read completion as follows: it indicates to the initiating agent on the north internal bus that it encountered an error by performing a target abort during the read completion. For a write transaction, the Bridge logs the target abort condition and completes the write transaction on the north internal bus.



6.3.7.2 Bridge South Internal Bus Interface Error

The following conditions may be encountered by the Bridge South Interface:

- Master Abort on the South Internal Bus interface. This condition may happen when the bridge attempts an address request on the south internal bus and the request is not claimed by any target. The bridge would log the error as a master abort.
 - For a write request, the bridge logs the master abort error condition and discard the write data.
 - For a read request, the Bridge logs the master abort. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the north internal bus that it encountered an error by performing a target abort during the read completion.
- Address Request Error on the south internal bus interface. This condition may happen when a target on the south internal bus detected an error during the address request. For example, the target may indicate that the byte-count is out of range or the target detected an address parity error.
 - For a write request, the bridge logs the address request error condition and discard the write data.
 - For read request, the Bridge logs the address request error. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the north internal bus that it encountered an error by performing a target abort during the read completion.
- Target Abort on the south internal bus. This condition may occur when a target claimed a previously issued read request, but is unable to return the read completion data. The bridge would log the error as a target abort.
 - The Bridge logs the target abort. In addition, the Bridge completes the read completion as follows: it indicates to the initiating agent on the north internal bus that it encountered an error by performing a target abort during the read completion. Address Parity Error on the south internal bus. This condition may occur when the Bridge south interface is acting as a target and either observes AERR asserted by another target on the south internal bus, or detects an address parity error while claiming an address request.



6.4 System Controller Register Definitions

The following registers are located in the Peripheral Memory-Mapped Register (PMMR) address space. They are accessible through the south internal bus accesses. The Internal Bus Arbitration Control Register provides controls for both the North and South Internal address busses. The south internal bus address and data test registers are used to force address or data parity errors on the south internal bus respectively. Note that the north internal bus does not support address and data parity.

- Internal Bus Arbitration Control Register
- South Internal Bus Address Test Register
- South Internal Bus Data Test Register
- Peripheral Memory-Mapped Register Base Address Register

The system controller only claims the address offset range +1640H through +164FH.



6.5 Internal Bus Bridge Register Definitions

The following registers are located in the Peripheral Memory-Mapped Register (PMMR) address space. They are only accessible from the south internal bus. Accesses to the Bridge registers that originate from the north internal bus are propagated to the south internal bus, and then claimed by the Bridge on the south interface. The Bridge Error Status register indicates the type of error that was encountered by the bridge on either the north or south interfaces. The Bridge Error Address and Error Upper Address registers indicate the address of the request that encountered the error. The Bridge Window Base Address and Window Limit Registers together define a memory window for the Bridge to claim transactions on the south internal bus.

- Bridge Window Base Address Register
- Bridge Window Upper Base Address Register
- Bridge Window Limit Register
- Bridge Error Status Register
- Bridge Error Address Register
- Bridge Error Upper Address Register

The internal bus bridge only claims the address offset range +1780H through +1797H.



6.5.1 Internal Bus Arbitration Control Register — IBACR

The 81341 and 81342 has two internal buses: the north internal bus and the south internal bus. Refer to the block diagram shown in [Figure 77, “Intel® 81341 and 81342 I/O Processors Block Diagram \(External\)”](#) on page 541. The two internal buses are identical and provide de-multiplexed address and data buses. Therefore, each internal bus consists of two independent arbiters: an **address** arbiter and a **data** arbiter. The north internal bus address arbiter controls the north internal bus **address** initiators, whereas the north internal data arbiter controls the north internal bus **data** initiators. Similarly, the south internal bus address arbiter controls the south internal bus **address** initiators, whereas the south internal data arbiter controls the south internal bus **data** initiators.

The IBACR can be used to enable or disable an internal bus **address** initiator from acquiring the internal address bus. Note that this register does not disable the agent. It only prevents an address initiator arbitrating for the internal address bus from acquiring the internal address bus. Bits [15:0] control the address initiators on the north internal address bus, and bits [31:16] control the address initiators on the south internal address bus.

Warning: Since the internal address arbiter parks on an agent that was last granted the internal address bus, disabling a parked agent using the IBACR does not take effect immediately. The IBACR disable bit is observed by the internal address arbiter only when an agent arbitrates for the internal address bus.

Note: The data initiators cannot be disabled.

Table 333. Internal Bus Arbitration Control Register — IBACR (Sheet 1 of 2)

Bit	Default	Description
31:25	00H	Reserved.
24	0 ₂	Reserved.
23	0 ₂	Reserved.
22	0 ₂	Reserved.
21	0 ₂	DDRMCU/MU Group Control — this bit controls the MU initiator. The MU address initiator is not granted the south internal address bus when this bit is set. 0 = Enabled 1 = Disabled Note: The LMU acts as an internal bus address initiator only for issuing MSI or MSI-X writes to the ATUE or ATUX. Otherwise, the MU is a target device on the internal bus, which are accessed by the Cores or indirectly by the PCI Host via the ATUs.
20	0 ₂	Reserved.



Table 333. Internal Bus Arbitration Control Register — IBACR (Sheet 2 of 2)

South XBG +1640H internal bus address offset Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
19	0 ₂	Application DMA Group Control — this bit controls the Application DMA Initiators. The ADMA address initiators are not granted the south internal address bus when this bit is set. 0 = Enabled 1 = Disabled
18	0 ₂	ATU-X Control — this bit controls the ATU-X. The ATU-X address initiator is not granted the south internal address bus when this bit is set. 0 = Enabled 1 = Disabled
17	0 ₂	ATU-E Control — this bit controls ATU-E. The ATU-E address initiator is not granted the south internal address bus when this bit is set. 0 = Enabled 1 = Disabled
16	0 ₂	SMBus Group Control — this bit controls the SMBus and PMON initiators. The SMBus and PMON address initiators are not granted the south internal address bus when this bit is set. 0 = Enabled 1 = Disabled
15:03	000H	Reserved.
02	0 ₂	Reserved.
01	0 ₂	XSC coreID1 control — this bit controls coreID1. The Intel XScale® processor with coreID1 address initiator is not granted the north internal address bus when this bit is set. 0 = Enabled 1 = Disabled
00	0 ₂	XSC coreID0 control — this bit controls coreID0. The Intel XScale® processor with coreID0 address initiator is not granted the north internal address bus when this bit is set. 0 = Enabled 1 = Disabled



6.5.2 South Internal Bus Address Test Control Register — SIBATCR

The SIBATCR can be used to inject an address parity error on the south internal address bus. The user must provide the ID of the initiator and also set the enable bit. The enable bit (when set) is used by hardware to inject an address parity error on the next address transaction provided the programmed ID matches. The parity error is injected in the address phase of the request. The enable bit is cleared by hardware in the cycle that follows the address phase. This is done to prevent recurring parity errors.

Note: In order to inject the address parity error in the desired transaction, the user must try to write this register immediately before the transaction is issued on the internal bus.

Table 334. South Internal Bus Address Test Control Register — SIBATCR

Bit	Default	Description
31:21	000H	Reserved.
20:16	00000 ₂	Address Parity Mask bits — Each bit of the generated address parity is XORed with the appropriate bits in this mask field before the parity bits are driven on the north internal bus. Bit16 corresponds to address parity bit 0, bit 17 corresponds to address parity bit 1, and so on.
07:04	0000 ₂	Initiator ID — This field specifies the initiator ID of the address (for example, the unit sourcing the address). Refer to Table 330, “Data Parity Testing Completer IDs” on page 540 for the initiator IDs.
03:01	000 ₂	Reserved.
00	0 ₂	Enable bit — When this bit is set, an address parity error is injected on the next address request made when the initiator ID of the transaction matches the specified Initiator ID in bits[07:04]. This bit is set by software and cleared by hardware.



The SIBDTCR can be used to inject a data parity error on the south internal data bus. The user must provide the ID of the initiator and also set the enable bit. The enable bit (when set) is used by hardware to inject a data parity error on the next data transaction provided the programmed ID matches. The parity error is injected in the first data phase. The enable bit is cleared by hardware in the cycle that follows the first data phase. This is done to prevent recurring parity errors.

Note: In order to inject the data parity error in the desired transaction, the user must try to write this register immediately before the transaction is issued on the internal bus.

Table 335. South Internal Bus Data Test Control Register — SIBDTCR

Intel® 81341 and 81342 I/O Processors
Developer's Manual
554

December 2007
Order Number: 315037-002US



6.5.4 Peripheral Memory-Mapped Register Base Address Register — PMMRBAR

This is a 32-bit register that contains the Base or starting address of the Peripheral Memory-Mapped Register space (PMMR). The PMMR space is aligned on a 512-KByte boundary. The PMMRBAR can be used to relocate the PMMR block to any 512-KByte space of the 64 GBytes address space. Bits [31:15] of this register represent bits [35:19] of the 36-bit internal bus address. The PMMR block default starting address after reset is 0 FFD8 0000H.

Table 336. Peripheral Memory-Mapped Register Base Address Register — PMMRBAR

		IOP Attributes PCI Attributes																															
		Intel XScale® processor Local Bus Address																Attribute Legend:															
South XBG		F FFFF FFFCH																RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible															

Bit	Default	Description
31:15	0000 1111 1111 1101 12	PMMR Base Address: These bits define the starting address of the PMMR block. The PMMR block is aligned on a 512-KByte boundary. These bits represent bits [35:19] of the 36-bit internal bus address.
14:00	000 0000 0000 0000 2	Reserved.



6.5.5 Determining Block Sizes for Memory Windows

The memory window size can be determined by writing ones to the appropriate upper bits of the limit register. The binary-weighted value of the first non-zero bit set in the limit register indicates the size of the memory window. Table 337 describes the relationship between limit register values and the byte sizes of the memory window.

Table 337. Memory Block Size Limit Register Value

Limit Register Value ^a	Size (in Bytes)	Limit Register Value	Size (in Bytes)
FFFFF000H	4K	FF000000H	16M
FFFFE000H	8K	FE000000H	32M
FFFFC000H	16K	FC000000H	64M
FFFF8000H	32K	F8000000H	128M
FFFF0000H	64K	F0000000H	256M
FFFE0000H	128K	E0000000H	512M
FFFC0000H	256K	C0000000H	1G
FFF80000H	512K	80000000H	2G
FFF00000H	1 M	00000000H	Address Window Closed.
FFE00000H	2 M		
FFC00000H	4 M		
FF800000H	8 M		

a. The smallest window supported is 4 Kbytes.

As an example, assume that FFF0 0004H is written to the Bridge Window Limit Register (BWLR). Scanning upwards starting at bit 12, bit 20 is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicating a 1 Mbyte of memory window.

When programming the Base and Limit Registers for a memory window, the Base Address always needs to be aligned the size of the memory window set in a limit register. For a 1 Mbyte memory window, only bit 20 through bit 31 of the base address from the Bridge Base Address Register (BBAR) are relevant to the Bridge when decoding Memory Window.

Warning: Care must be exercised when modifying the Bridge Base (BBAR) and Limit (BWLR) register pair during the time there is activity.

Warning: Since the smallest bridge memory window is 4 KBytes, care must be exercised when the bridge memory window is programmed to access memory-mapped registers that are located on the north internal bus. The 4-KByte bridge window overlaps other south internal bus memory-mapped registers. The user must not access the south internal bus memory-mapped registers that overlap the bridge memory window until the bridge memory window is re-mapped and does not overlap.



6.5.6 Bridge Window Base Address Register — BWBAR

The Bridge Base Address Register (BWBAR) defines the block of memory addresses where the Bridge Memory Window begins. The BWBAR is used in conjunction with the BWLR to form a memory window that is used by the Bridge to claim transactions on the South Internal Bus. The BWBAR defines the base address and describes the required memory block size; see [Section 6.5.5, “Determining Block Sizes for Memory Windows” on page 556](#). The selected base address needs to be naturally aligned to the granularity of the memory block size. For instance, when a 64 Kbyte memory window size is selected, the base address needs to be 64 Kbyte address aligned (i.e., bits 15:12 of the base address are required to be 000b).

Table 338. Bridge Window Base Address Register — BWBAR

Diagram illustrating the Bridge Memory Window Base Address (bits 31:0) and its attributes:

- IOP Attributes:** RW (Read/Write), RV (Reserved), RC (Read Clear), RO (Read Only), RS (Read/Set).
- PCI Attributes:** NA (Not Accessible).

The diagram shows the address bits 31:0, with the first 12 bits (31:0) being IOP attributes and the last 4 bits (3:0) being PCI attributes. The attributes are mapped to the address bits as follows:

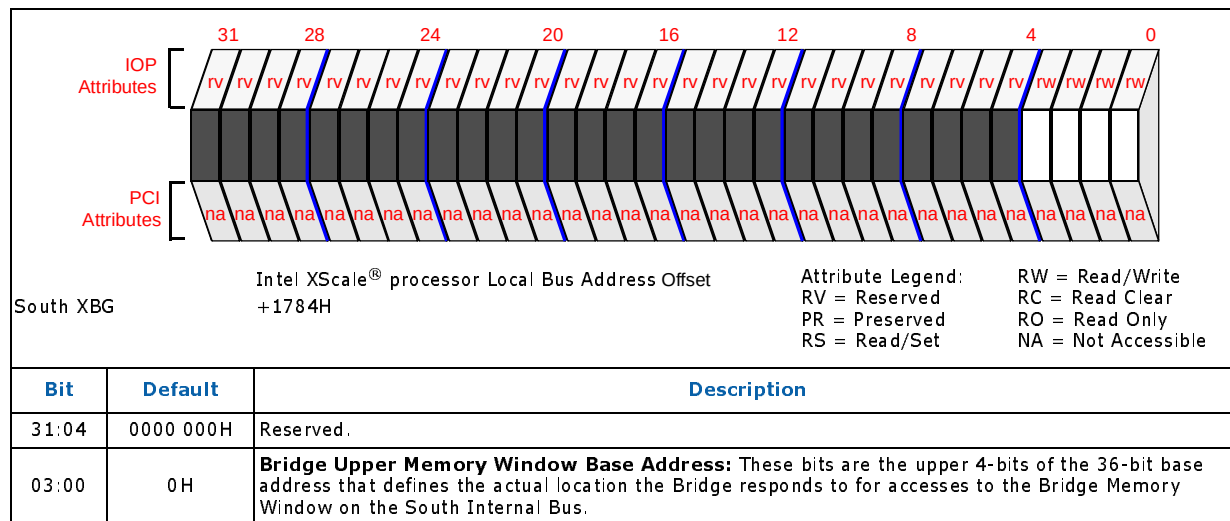
- Bits 31:28: RW
- Bits 27:24: RW
- Bits 23:20: RW
- Bits 19:16: RW
- Bits 15:12: RW
- Bits 11:8: RV
- Bits 7:4: RV
- Bits 3:0: RV

Legend:

- RW = Read/Write
- RV = Reserved
- RC = Read Clear
- RO = Read Only
- RS = Read/Set
- NA = Not Accessible



The Bridge Window Upper Base Address Register (BWUBAR) provides the upper 4 bits of the block of memory addresses where the Bridge Memory Window begins. The BWUBAR is used in conjunction with the BWBAR to form a 36-bit base address register. Refer to the [Section 6.5.6, “Bridge Window Base Address Register — BWBAR” on page 557](#).





6.5.8 Bridge Window Limit Register — BWLR

The 81341 and 81342 limit register's (BWLR) programmed value must be naturally aligned with the base address register's (BWBAR) programmed value. The limit register is used as a mask when the address decode for Bridge memory window is performed.

Table 340. Bridge Limit Register — BWLR

		South XBG Intel XScale® processor Local Bus Address Offset +1788H Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible																																			
Bit	Default	Description																																			
31:12	FFF0 0H	Bridge Memory Window Limit: This value determines the memory range required for the Bridge Memory Window. Defaults to a 1MB Memory Window.																																			
11:00	000H	Reserved																																			



6.5.9 Bridge Error Control and Status Register — BECSR

The BECSR logs the type of error that the bridge encountered on either north or south interface. Only one error can be logged at a time.

The Bridge has two interrupt conditions: first Bridge error (BECSR[0]), and more than one Bridge error (BECSR[1]).

When the Bridge detects an error and BECSR[0] is cleared, the error is logged in BECSR, and BECSR[0] is set to 1. When BECSR[0] is not cleared, any additional Bridge errors are not logged and BECSR[1] is set.

Table 341. Bridge Error Control and Status Register — BECSR (Sheet 1 of 2)

Bit	Default	Description
31:17	0000H	Reserved.
16	0 ₂	Error Reporting Enable: This bit when set enables the internal bus bridge to signal an interrupt to the Intel XScale® processor when a South Internal Bus Error is detected. When cleared, an interrupt to the Intel XScale® processor is not signaled when a South Internal Bus Address Error is detected. Note: This enable applies to all error types reported by the Internal Bus bridge through BECSR[3:2]
15:12	0H	Reserved.
11:08	0H	This field logs the Requester ID of the transaction that resulted in an error. Requester ID Core 0 1 Core 1 2 ATU-X 3 ATU-E 4 ADMA 5 N/A 6 LMU 7 Reserved 8 SMBus 9
07:06	00 ₂	Reserved.
05	0 ₂	Interface Type: This bit indicates the Bridge interface that encountered the error condition. 0 = North Internal Bus Interface. 1 = South Internal Bus Interface
04	0 ₂	Command Type: Indicates the command type that encountered the error: 0 = Read. 1 = Write.



Table 341. Bridge Error Control and Status Register — BECSR (Sheet 2 of 2)

The diagram illustrates the South XBG internal bus address offset and attribute legend. It shows a 32-bit address space (0 to 31) divided into two main sections: IOP Attributes (top) and PCI Attributes (bottom). The address is split into four 8-bit segments (0-7, 8-15, 16-23, 24-31). The IOP Attributes section is divided into four 4-bit segments (0-3, 4-7, 8-11, 12-15). The PCI Attributes section is divided into four 4-bit segments (0-3, 4-7, 8-11, 12-15). The attribute legend defines the following values:

- RW = Read/Write
- RV = Read/Valid
- PR = Preserved
- RS = Read/Set
- RO = Read Only
- NA = Not Accessible

The diagram shows the following attribute values for the 32-bit address space:

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



6.5.10 Bridge Error Address Register — BERAR

This register is responsible for logging the lower 32-bit of the 36-bit address of the bridge request that encountered the error. This register is used in conjunction with the BERUAR. Refer to [Section 6.5.11, “Bridge Error Upper Address Register — BERUAR”](#) on page 562.

Table 342. Bridge Error Address Register — BERAR

Bit	Default	Description																											
31:00	0000 0000H	Error Address: Stores the lower 32-bit of the 36-bit address of the request that resulted in an error.																											

6.5.11 Bridge Error Upper Address Register — BERUAR

This register is responsible for logging the upper 4-bit address of the 36-bit address of the Bridge request that encountered the error. This register is used in conjunction with the BERAR ([Section 6.5.10, “Bridge Error Address Register — BERAR”](#) on page 562).

Table 343. Bridge Error Upper Address Register — BERUAR

Bit	Default	Description																											
31:04	0000 000H	Reserved.																											
03:00	0000 ₂	Error Address: Stores the upper 4 bits of the 36-bit address of the request that resulted in an error.																											



7.0 DDR SDRAM Memory Controller

This chapter describes the integrated Memory Controller Unit (DMCU), with operating modes, initialization, external interfaces, and implementation details.

7.1 Overview

The Intel® 81341 and 81342 I/O Processors (81341 and 81342) integrates a high-performance, multi-ported Memory Controller to provide a direct interface between the 81341 and 81342 and its local memory subsystem. The Memory Controller supports:

- PC3200/PC4300 Double Data Rate II (DDR2 400 MHz and DDR2 533 MHz) SDRAM
- 512 Mbit and 1 Gbit and 2 Gbit DDR-II SDRAM technology support
- Registered and Unbuffered DDR2 DIMM support
- Dedicated port for Intel XScale® microarchitectures to DDR2 SDRAM
- Between 256 MBytes and 4 GBytes of 64-bit DDR2 SDRAM
- 36-bit addressable
- Two memory windows to address the same DDR SDRAM Memory
- Optimized core processor data processing 32-bit region
- Single-bit error correction, multi-bit detection support (ECC)
- 32-, 40- and 64-, 72-bit wide Memory Interfaces (non-ECC and ECC support)

DDR2 SDRAM interface provides a direct connection to a high bandwidth and reliable memory subsystem. The DDR2 SDRAM interface consists of a 64-bit wide data path to support up to 4.3 GBytes/s throughput. An 8-bit Error Correction Code (ECC) across each 64-bit word improves system reliability. The ECC is stored in the DDR SDRAM array along with data and checked when data is read. When code is incorrect, the DMCU corrects data (when possible) before reaching the read initiator. User-defined fault correction software is responsible for scrubbing the memory array. DMCU supports two banks of DDR SDRAM made up of a two-bank dual-inline memory module (DIMM):

- DMCU has support for Registered and Unbuffered PC3200 and PC4300 DIMMs.
- The DMCU supports a 32-bit SDRAM data interface. This mode enables lower-cost solutions at the cost of system performance.
- The DMCU responds to internal bus and core processor memory accesses within its programmed address range and issues the memory request to the DDR SDRAM interface.
- The DMCU contains transaction queues for each port enabling pipelining of transactions to the DDR SDRAM for maximum performance.
- For 64-bit ECC memory, a 32-bit memory region programmable to operate as 32-bit ECC memory for higher core write performance by avoiding Read-Modify-Write (RMW) operation of DDR SDRAM.
- The DMCU provides two chip enables to the memory subsystem. These two chip enables service the DDR SDRAM subsystem (one per bank).



7.2 Glossary

This section lists commonly used terms throughout this chapter:

Table 344. Commonly Used Terms

Term	Definition
Bank	A bank is defined as a memory region defined with a base register and a bank size register. Physically, a bank of memory is controlled by a single chip select. A DIMM comprised of a single or dual banks.
Column	A column refers to a portion of memory within a DDR SDRAM device. A DDR SDRAM device is thought of as a grid with rows and columns. Once a row is activated, any column within that row is accessed multiple times without reactivating the row. Columns are activated with CAS# .
DIMM	A DIMM is an acronym for Dual Inline Memory Module. A DIMM is a physical card comprising multiple DDR SDRAM devices. The card is populated on one or both sides. A DIMM is single or dual-bank.
Leaf	DDR SDRAM devices use multiple banks within the device operating in an interleaved mode. The DMCU supports 512 Mbit and 1 Gbit and 2 Gbit DDR-II SDRAM devices. 512 Mbit DDR-2 SDRAM devices support four internal banks, whereas 1 Gbit and 2 Gbit DDR-2 SDRAM devices support eight internal banks. An internal bank is defined as a leaf (to avoid confusion with a memory bank).
Page	A page is a row of memory. Once a row is activated, any column within that row can be accessed multiple times without reactivating the row. This is referred to as "keeping the page open."
Row	A row refers to a portion of memory within a DDR SDRAM device. A DDR SDRAM device is thought of as a grid with rows and columns. Once a row is activated, any column within that row is accessed multiple times without reactivating the row. Rows are activated with RAS# .
Scrubbing	Once an error is detected within the memory array, the DMCU must correct the error (when possible) while delivering the data to the initiator. Correcting the memory location is referred to as "scrubbing the array." The DMCU relies on software to scrub any errors.
Syndrome	A syndrome is a value which indicates an error in the data read from the memory array. The DMCU computes the syndrome with every memory read. Decoding the syndrome indicates: the bit in error for a single-bit error, or a multi-bit error. Table 370 defines the syndrome decoding.



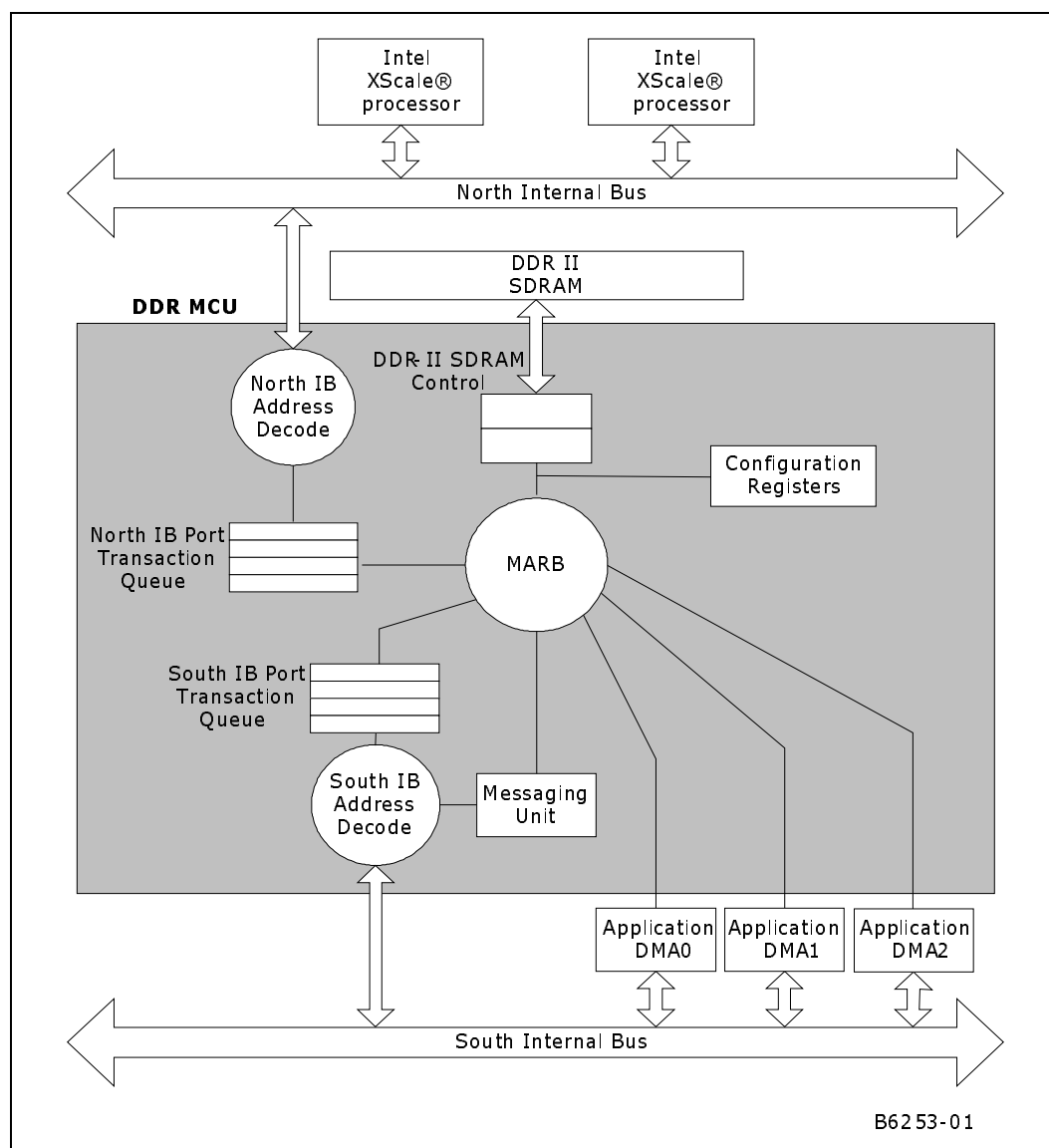
7.3 Theory of Operation

The 81341 and 81342 memory controller translates transactions from the North internal bus, South internal bus, the Application DMAs (ADMA), and the Messaging Unit into the protocol supported by the DDR2 SDRAM memory subsystem.

7.3.1 Functional Blocks

The memory controller logically comprises the blocks illustrated in Figure 78. The memory controller is a multi-ported unit, supporting inbound paths from the north internal bus, the south internal bus, the Application DMAs to the DDR2 SDRAM.

Figure 78. Memory Controller Block Diagram





7.3.1.1 Transaction Ports

The DMCU provides six transaction ports for direct DDR SDRAM access. The ports are described in the following sub-sections.

7.3.1.1.1 North Internal Bus Port

The North IB Port provides the 81341 and 81342 processor core access to the Memory Controller. This North IB Port allows core transactions targeting the DDR SDRAM via the North IB bus to pass directly to the DDR SDRAM.

7.3.1.1.2 South Internal Bus Port

The South IB Port provides the connection to the DDR SDRAM from the South IB Bus. Peripheral unit transactions targeting the DDR SDRAM are claimed by this port. For example, the ATU transactions are claimed by the South IB port. Note that the Application DMAs connect to the DDR SDRAM directly using other DMCU ports.

7.3.1.1.3 Application DMA Ports

The Application DMAs connect to the DMCU using dedicated ports — Application DMA ports. There are three Application DMAs. The Application DMA can transfer data between DDR SDRAM or between DDR SDRAM and South IB agents.

7.3.1.1.4 Messaging Unit Port

The Messaging Unit (MU) connects to the DMCU using a dedicated port. From an application point of view, the MU is used in conjunction with the ATU.



7.3.1.2 Address Decode Blocks

Address Decode is performed for transactions from input North and South internal bus ports to determine when the DMCU claims the transaction. The North internal bus port claims transactions targeting the DDR SDRAM memory array space. The South internal bus port claims transactions for two address ranges:

- DDR SDRAM memory array space
- Peripheral Memory-Mapped Register (PMMR) space

7.3.1.2.1 DDR SDRAM Memory Array Space

The DDR SDRAM Memory Array Space is accessed from either the North internal bus port, South internal bus port, or the Application DMA ports using a Primary Memory Window. The primary DDR SDRAM memory space is defined with the DDR SDRAM Base Address Register (SDBR and SDUBR), the DDR SDRAM Bank Size Register (SDSR) and the SDRAM Control 1 register (SDCR1). The transaction is intended for a DDR SDRAM banks when the address is between the base registers (SDBR and SDUBR) and between the boundaries programmed with SBSR and S32SR as defined in [Section 7.3.3.3, "DDR SDRAM Bank Sizes and Configurations" on page 578](#).

In addition, the Intel XScale® core can access part of the DDR SDRAM Memory Array from the North internal bus port using a Secondary Memory Window. The Secondary Memory Window is used in conjunction with the Primary Memory Window. The Secondary Memory Window is setup using the Secondary SDRAM Base Register (SSDBR) and the DDR SDRAM Bank Size Register (SBSR).

7.3.1.2.2 Memory-Mapped Register Space

The DMCU PMMR memory space is at offsets +1800H through +19FFH. The registers are detailed in [Section 7.8, "Register Definitions" on page 625](#).

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.

7.3.1.2.3 North Internal Bus Port Address Decode

North internal bus port transactions are decoded to determine when they address the DDR Memory space. When the transaction addresses the DDR SDRAM Memory Space, the transaction is queued in the North internal bus port queue.

7.3.1.2.4 South Internal Bus Port Address Decode

South internal bus port transactions are decoded to determine when they address the DDR SDRAM Memory Space, the DMCU MMR Space, or the Messaging Unit Space. When the transaction addresses either of these spaces, the transaction is claimed by the South Internal Bus Port. When the transaction addresses the DDR SDRAM Memory Space, the transaction is queued in the south internal bus Port transaction queue. When the transaction addresses the DMCU MMR Space, the transaction is serviced by accessing the Configuration Register block.

7.3.1.2.5 Application DMA Port Address Decode

The Application DMA Ports do not require any decode, and processes any transaction received from the Application DMAs via the Application DMA DMCU Port.



7.3.1.3 Memory Transaction Queues

There are two sets of transaction queues for transactions which address the DMCU DDR Memory Space: transaction queue for north internal bus transactions and transaction queue for south internal bus transactions. The dedicated units that are connected to the DMCU do not support transaction queues.

7.3.1.3.1 North Internal Bus Port Transaction Queue (NIBPTQ)

The NIBPTQ stores memory transactions from the north internal bus. The NIBPTQ supports 16 read transactions, each with up to 32 bytes buffer. The NIBPTQ also supports 16 posted write transactions up to 32 Bytes each.

7.3.1.3.2 South Internal Bus Port Transaction Queue (SIBPTQ)

The SIBPTQ stores memory transactions from the south internal bus. The SIBPTQ supports 8 read transactions, each with up to 1024 bytes buffer. The SIBPTQ also supports 8 posted write transactions up to 1024 Bytes each.

7.3.1.4 Configuration Registers

The Configuration Registers block contains all of the memory-mapped registers listed in [Section 7.8, "Register Definitions" on page 625](#). These registers define the memory subsystem connected to the 81341 and 81342. The status registers indicate the current DMCU status.

7.3.1.5 Refresh Counter

The Refresh Counter block keeps track of when the DDR SDRAM devices need to be refreshed. The refresh interval is programmed in the ["Frequency Register — RFR" on page 651](#). Once the 12-bit refresh counter reaches the programmed interval, the DDR SDRAM state machine issues a refresh command to the DDR SDRAM devices. When a transaction is currently in progress, the DDR SDRAM State Machine waits for the completion of the transaction to issue the refresh cycle. See [Section 7.3.3.11, "DDR SDRAM Refresh Cycle" on page 606](#) for more details.

Note: When the memory controller is completing transactions from the north internal bus transaction queue (NIBPTQ) when the refresh counter expires, the DMARB completes the NIBPTQ tenure based on the ["DMCU Port Transaction Count Register — DMPTCR" on page 648](#), before any refresh cycles are issued.

Note: When the memory interface is busy when the refresh counter expires, it is possible for the DMCU to generate more than one refresh cycle when the memory interface becomes available.

7.3.1.6 Memory Controller Arbiter (DMARB)

The DMARB determines what transaction to issue next to the DDR SDRAM Control Unit. The DMARB is configurable to adjust the selection of transactions from the DMCU ports and the refresh counter. The DMARB selects transactions from all memory ports based on a programmable arbitration scheme as described in [Section 7.3.2, "DMCU Arbitration and Configuration" on page 570](#).



7.3.1.7 DDR SDRAM Control Block

The DDR SDRAM Control Block contains all functionality to process the DDR SDRAM data accesses per the transactions issued by the DMARB. To process a transaction the DDR SDRAM Control Block employs several sub-blocks. The sub-blocks include the Page Control block, DDR SDRAM State Machine and Pipeline Queues, and Error Correction Logic.

7.3.1.7.1 Page Control Block

The Page Control Block records and maintains the open DDR SDRAM pages. The DMCU can keep a maximum of eight/sixteen pages open simultaneously (4 or 8 per bank depending on the memory technology). 81341 and 81342 supports up to two memory banks, and 1-Gbit/2-Gbit DDR2 memory technology supports up to eight internal banks. This accounts for a total of sixteen pages when both memory banks are implemented. 512-bit DDR2 technology supports only four internal banks, which therefore accounts for a total of eight pages when both memory banks are implemented. This block keeps track of open pages and determines when the transactions hit an open page. For more details about the page hit/miss determination, see [Section 7.3.3.5, "Page Hit/Miss Determination" on page 589](#).

7.3.1.7.2 DDR SDRAM State Machine and Pipeline Queues

Since the DMCU generates error correction codes based on the data, the DMCU is a pipelined architecture. Pipelining also ensures acceptable AC timings to the memory interfaces. The DDR SDRAM state machine pipelines DDR SDRAM memory operations for several clocks.

7.3.1.7.3 Error Correction Logic

The Error Correction Logic generates the ECC code for DDR SDRAM reads and writes. For reads, this logic compares the ECC codes read with the locally generated ECC code. When 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 7.3.4, "DDR Error Correction and Detection" on page 607](#) for more details.



7.3.2 DMCU Arbitration and Configuration

The order transactions are processed by the DDR SDRAM Control block is determined by the Memory Controller Arbiter (DMARB). The DMARB selects the next transaction from the memory transaction queues and refresh control based on the settings programmed in the DMARB configuration registers “[DMCU Port Transaction Count Register — DMPTCR](#)” on [page 648](#). These registers enable the DMCU to be tuned for optimal design performance.

7.3.2.1 DMCU Port Priority

The Memory controller for the 81341 and 81342 has six inbound ports for memory transactions.

7.3.2.2 DMCU Port Transaction Count

The DMCU Port Transaction Count Register (DMPTCR) defines the number of transactions the DMARB selects from a given port during the corresponding port tenure. The intent is that ports with frequent small transactions (north internal bus port) be allowed to complete multiple transactions during a tenure, while ports with less frequent large transactions be limited to a small number of transactions (possibly just 1).



7.3.2.3 North Internal Bus Port Preemption

In order to maximize core processor performance, the DMARB can be programmed to interrupt active transactions from other memory ports, in order to process transactions from the north internal bus (Core Processors). By preempting an active transaction, the DMARB ensures a maximum number of data cycles getting processed for the current transaction, therefore reducing the possible latency the Core Processor might see for memory accesses. Core processor preemption is enabled through the DMARB Preemption Control Register (DMPCR).

Preemption control applies to all inbound DMCU ports based on pending north internal bus transactions. Once the north internal bus transaction queue is 'activated' by the DMARB, a standard tenure takes place. At the completion of the north internal bus transaction queue tenure, the DMARB resumes the previously active transaction. The previously active transaction may be interrupted again, provided the conditions for preemption occur again prior to the completion of that transaction. No other effect on the DMARB processing occurs.

7.3.2.4 North Internal Bus Port Transaction Ordering

The North IB port maintains order of requests (core requests) addressing the DDR SDRAM. Coherency between the north IB port, south IB port and the other ports are maintained by the DMCU as described in "[DMCU Port Coherency](#)" below.

7.3.2.5 South Internal Bus Port Ordering

The IB port implements ordering of like transactions where reads do not pass reads, and writes do not pass writes. Write transactions are allowed to pass read transactions. Read transactions are processed by the DMCU as described in "[DMCU Port Coherency](#)" below.

7.3.2.6 Application DMAs Port Ordering

The Application DMA port implements ordering of like transactions where reads do not pass reads, and writes do not pass writes. Write transactions are allowed to pass read transactions. Read transactions are processed by the DMCU as described in "[DMCU Port Coherency](#)" below.

7.3.2.7 DMCU Port Coherency

With the queueing of DDR SDRAM transactions in multiple ports, coherency of memory must be maintained. The DMARB maintains memory coherency by ensuring that all writes to a given memory address are completed before any read to the same address is processed. This address comparison is done with a 1 KByte granularity.

The highest priority Read transaction is compared to all pending write transactions from all memory transaction queues (north IB and south IB port queues). When a write transaction is pending for the same memory location (1 KByte granularity), the write is allowed to complete first, before the read transaction is processed. Also, to maintain ordering rules, all write transactions preceding the 'incoherent write' are also processed.



7.3.3 DDR SDRAM Memory Support

The 81341 and 81342 memory controller supports one or two banks of DDR SDRAM. DDR SDRAM allows zero data-to-data wait-state operation. DDR SDRAM offers an extremely wide range of configuration options emerging from the SDRAMs internal interleaving and bursting capabilities.

The DMCU supports both 32-bit and 64-bit data bus width memory implementations (with and without ECC). The data bus width is controlled by the DDR SDRAM Control Register. In addition, a 64-bit data bus width DDR SDRAM implementation is configured to operate as when a region is 32-bits wide, providing higher performance when the core processor is processing data by eliminating any RMW cycle required for 4-Byte store ECC generation.

7.3.3.1 DDR SDRAM Interface

Table 346 shows the DDR2 SDRAM interface maximum bandwidth.

Table 345. DDR SDRAM Memory Configuration Options

Data Bus Width	ECC Enabled	Maximum Throughput ^a
64 bit	Yes	4264 Mbyte/s
64 bit	No	4264 Mbyte/s
32 bit	Yes	2132 Mbyte/s
32 bit	No	2132 Mbyte/s

a. Based on DDR2 533MHz SDRAM.

**Table 346. DDR2 SDRAM Interface Signals**

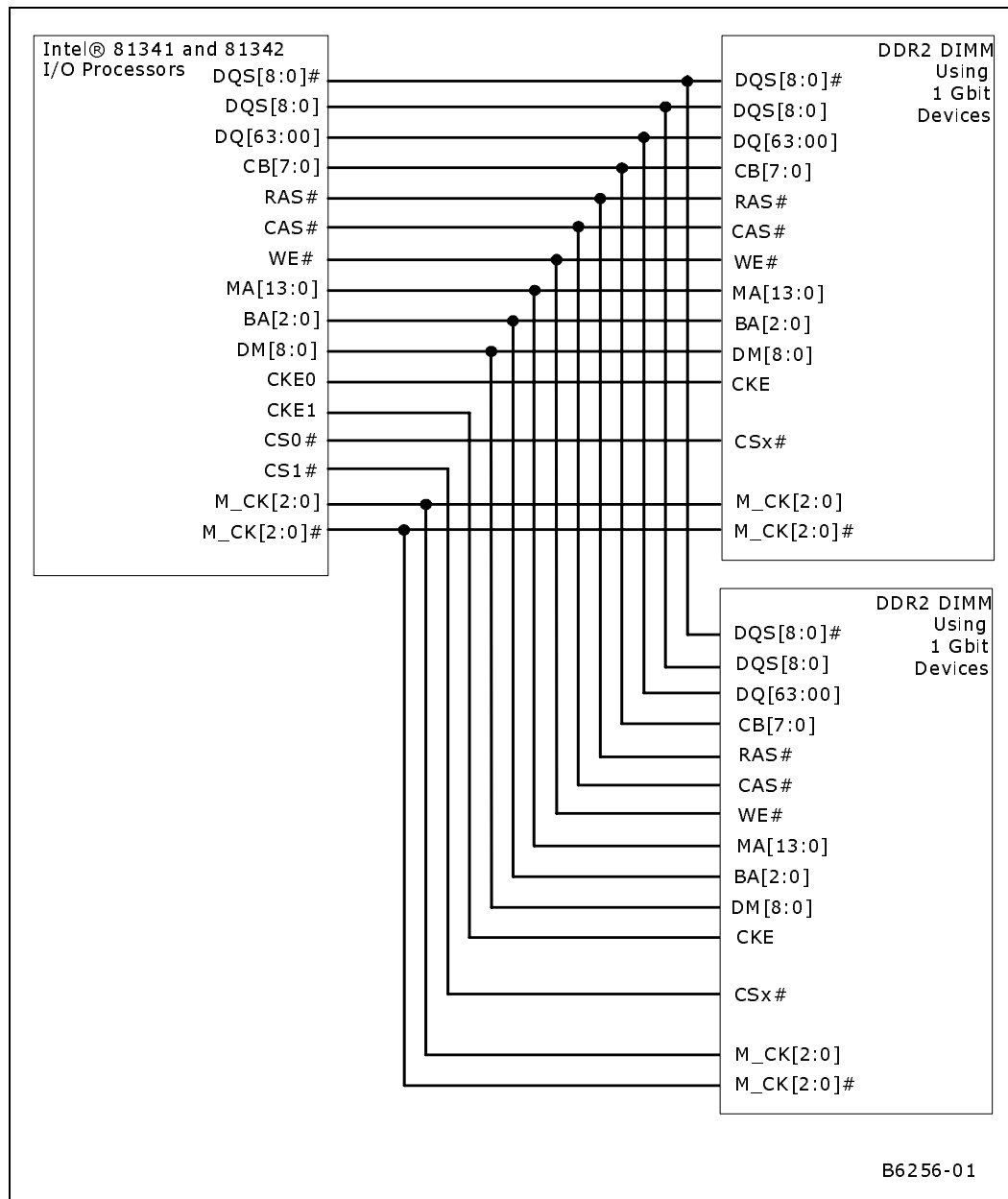
Pin Name	Description
M_CK[2:0]	<i>DDR SDRAM Clock Out</i> — Three (positive lines) output clocks driven to Unbuffered DIMMs supported by the 81341 and 81342. Registered DIMMs only use M_CK[0] which drives the input to the on-DIMM PLL. Section 7.3.6, “DDR SDRAM Clocking” on page 616 describes the DDR SDRAM clocking strategy for both Unbuffered and Registered DIMMs.
M_CK[2:0]#	<i>DDR SDRAM Clock Out</i> — Three (negative lines) output clocks driven to Unbuffered DIMMs supported by the 81341 and 81342. Registered DIMMs only use M_CK[0]# which drives the input to the on-DIMM PLL. Section 7.3.6, “DDR SDRAM Clocking” on page 616 describes the DDR SDRAM clocking strategy for both Unbuffered and Registered DIMMs.
M_RST#	<i>DDR Registered DIMM Reset</i> — Re-initializes registered DIMM during a P_RST# assertion or whenever internal bus reset bit is asserted in the PCSR. Section 3.16.41, “PCI Configuration and Status Register - PCSR” on page 331 describes internal bus reset sequence.
CKE[1:0]	<i>Clock enables</i> — One clock after CKE[1:0] is de-asserted, data is latched on DQ[63:0] and CB[7:0] . Burst counters within DDR SDRAM device are not incremented. De-asserting this signal places DDR SDRAM in self-refresh mode. For normal operation, CKE[1:0] must be asserted.
CS[1:0]#	<i>Chip Select</i> — Must be asserted for all transactions to DDR SDRAM device. One per bank.
BA[2:0]	<i>DDR SDRAM Bank Selects</i> — Controls which of the internal DDR SDRAM banks to read or write. Only BA[1:0] are used for 512 Mbit technology type, whereas BA[2:0] are used for 1 Gbit and 2 Gbit technology types. Note that 81341 and 81342 supports only DDR2 512 Mbit and 1 Gbit and 2 Gbit technology types.
MA[10]	<i>Address bit 10</i> — When high during a read or write command, auto-precharge occurs after the command. During a row-activate command, this bit is part of the address (see Table 357). Auto-precharge is not supported by the 81341 and 81342.
MA[13:0]	<i>Address bits 13 through 0</i> — Indicates the row or column to access depending on the state of RAS# and CAS# (see Table 357).
DQ[63:0]	<i>Data Bus</i> — 64-bit wide data bus.
CB[7:0]	<i>ECC Bus</i> — 8-bit error correction code which accompanies the data on DQ[63:0] .
DM[8:0]	<i>Data Bus Mask</i> — Controls the DDR SDRAM data input buffers. Asserting WE# causes the data on DQ[63:0] and CB[7:0] to be written into the DDR SDRAM devices. DM[8:0] controls this operation on a per byte basis.
DQS[8:0], DQS[8:0]#	<i>Data Strokes</i> — carry the differential strobe signals, output in write mode and input in read mode for source synchronous data transfer.
WE#	<i>Write Strobe</i> — Defines whether or not the current operation by the DDR SDRAM is to be a read or a write.
RAS#	<i>Row Address Strobe</i> — Indicates that the current address on MA[13:0] is the row.
CAS#	<i>Column Address Strobe</i> — Indicates that the current address on MA[13:0] is the column.
ODT[1:0]	<i>On Die Termination signals control</i> — turns on SDRAM termination during writes
M_VREF	<i>SDRAM Voltage Reference</i> — is used to supply the reference voltage to the differential inputs of the memory controller pins.
MCAL[1:0]	<i>Memory Calibration</i> — Connected to an external calibration resistor. Memory output drivers can reference the resistor to dynamically adjust slew rate and drive strength to compensate for temperature and voltage variations. M_CAL[0] is connected through a 25-ohm 1% resistor to ground. M_CAL[1] is connected through a 300-ohm 1% resistor to ground.

Utilizing the DDR SDRAM chip selects **CS[1:0]#** and internal bank selects **BA[2:0]**, the DMCU keeps a maximum of eight pages open simultaneously for 512 Mbit DDR-2 SDRAM devices and sixteen pages for 1Gbit or 2 Gbit DDR-2 SDRAM devices. The number of available pages depends on the memory subsystem population. A single 512 Mbit DDR-2 SDRAM device allows four pages, whereas a 1 Gbit or 2 Gbit DDR-2 SDRAM device allows eight pages.

Open pages allow optimal performance when a read or write occurs to an open page. Multiple open pages allow multiple memory segments to be open simultaneously and is well-suited for the 81341 and 81342 system environment. The DMCUs paging algorithm is detailed in [Section 7.3.3.5, “Page Hit/Miss Determination” on page 589](#). The waveforms illustrating the performance issues are in [Section 7.3.3.9, “DDR SDRAM Read Cycle” on page 603](#) and [Section 7.3.3.10, “DDR SDRAM Write Cycle” on page 604](#).

Figure 79 illustrates how two banks of DDR SDRAM interface with the 81341 and 81342 through the DMCU.

Figure 79. Intel® 81341 and 81342 I/O Processors Dual-Bank DDR SDRAM Memory Subsystem





7.3.3.2 DDR SDRAM Addressing

The internal bus address is mapped differently to the Bank Address/Row Address/Column Address of the DDR2 memory based on the setting of bit 6 of SDCR0. Refer to “DDR SDRAM Control Register 0 — SDCR0” on page 628 for more details.

Table 347 and Table 348 illustrate how the internal address is mapped to the MA[13:0] lines when bit 6 of SDCR0 is cleared.

Table 347. DDR2 SDRAM Address Translation for 512 Mbit (x16) and 1 Gbit (x16) Devices (SDCR0[6] cleared)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	-	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	A[13]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 357.

Table 348. DDR2 SDRAM Address Translation for 512 Mbit (x8), and 1 Gbit (x8) Devices (SDCR0[6] cleared)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	A[13]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 357.

Table 349. DDR2 SDRAM Address Translation for 2 Gbit (x16) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	A[13]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

Table 350. DDR2 SDRAM Address Translation for 2 Gbit (x8) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]	A[14]	A[13]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.



Table 351 and Table 352 illustrate how the internal address is mapped to the MA[13:0] lines when the SDCR0 bit 6 is set.

Table 351. DDR2 SDRAM Address Translation for 512 Mbit (x16) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	-	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

Table 352. DDR2 SDRAM Address Translation for 1 Gbit (x16) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	-	A[28]	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

Table 353. DDR2 SDRAM Address Translation for 512 Mbit (x8) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	A[28]	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]	A[15]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

Table 354. DDR2 SDRAM Address Translation for 1 Gbit (x8) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	A[29]	A[28]	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

Table 355. DDR2 SDRAM Address Translation for 2 Gbit (x16) Device (SDCR0[6] set)

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	-	A[29]	A[28]	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See Table 364 for more details.
2. For the Leaf Selects, see Table 358.

**Table 356. DDR2 SDRAM Address Translation for 2 Gbit (x8) Device (SDCR0[6] set)**

MA[14:0]	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	A[30]	A[29]	A[28]	A[27]	A[26]	A[25]	A[24]	A[23]	A[22]	A[21]	A[20]	A[19]	A[18]	A[17]	A[16]
Column	-	-	-	-	V ¹	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]

1. A10 is used for precharge variations on the read or write command. See [Table 364](#) for more details.
2. For the Leaf Selects, see [Table 358](#).

Since the DMCU supports DDR SDRAM bursting, the DMCU increments the column address based on burst length of four for each DDR SDRAM read or write burst. The DMCU supports sequential and random burst types. Sequential bursting means the address issued to DDR SDRAM is incremented by the DDR SDRAM device in linear fashion during burst cycle. Random bursting means that the address issued with every new command to the DDR SDRAM can be any address in a currently active page.

Note: The DMCU only supports Sequential Burst Type. For example, Interleaved Burst Type is not supported by the DMCU.



7.3.3.3 DDR SDRAM Bank Sizes and Configurations

The DMCU supports a memory subsystem ranging from 128 Mbytes to 4 GBytes. An ECC or non-ECC system is implemented using x8, or x16 devices. [Table 357](#) illustrates supported DDR SDRAM configurations when bit 6 of SDCR0 is cleared, whereas [Table 358](#) illustrates supported DDR SDRAM configurations when bit 6 of SDCR0 is set.

Note: 128 Mbytes DDR memory size is only applicable when the DMCU is setup to 32-bit data bus width. Otherwise, the minimum DDR2 memory that can be supported with a 64-bit data bus is 256 MBytes for one bank.

Table 357. Supported DDR2 SDRAM Configurations (SDCR0[6] cleared)

DDR2 SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select			Total Memory Size ^a
			Row	Column	BA[2]	BA[1]	BA[0]	
512 Mbit	64Mb x 8	1	14	10	n/a	A[28]	A[27]	512M
		2						1G
	32Mb x 16	1	13	10	n/a	A[27]	A[26]	256M
		2						512M
1 Gbit	128Mb x 8	1	14	10	A[29]	A[29]	A[27]	1G
		2						2G
	64Mb x 16	1	13	10	A[28]	A[27]	A[26]	512M
		2						1G
2 Gbit	256Mb x 8	1	15	10	A[30]	A[29]	A[28]	2G
		2						4
	128Mb x 16	1	14	10	A[29]	A[28]	A[27]	1G
		2						2G

a. Table indicates 64-bit wide memory subsystem sizes. For 32-bit wide memory, the memory subsystem size would be half of that indicated.

Table 358. Supported DDR2 SDRAM Configurations (SDCR0[6] set)

DDR2 SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select			Total Memory Size ^a
			Row	Column	BA[2]	BA[1]	BA[0]	
512 Mbit	64Mb x 8	1	14	10	n/a	A[14]	A[13]	512M
		2						1G
	32Mb x 16	1	13	10	n/a	A[14]	A[13]	256M
		2						512M
1 Gbit	128Mb x 8	1	14	10	A[15]	A[14]	A[13]	1G
		2						2G
	64Mb x 16	1	13	10	A[15]	A[14]	A[13]	512M
		2						1G
2 Gbit	256Mb x 8	1	15	10	A[15]	A[14]	A[13]	2G
		2						4
	128Mb x 16	1	14	10	A[15]	A[14]	A[13]	1G
		2						2G

a. Table indicates 64-bit wide memory subsystem sizes. For 32-bit wide memory, the memory subsystem size would be half of that indicated.



512 Mbit DDR2 SDRAM devices comprise four internal leaves. The DMCU controls the leaf selects within 512 Mbit DDR2 SDRAM by toggling **BA[0]** and **BA[1]**.

1 Gbit and 2 Gbit DDR2 SDRAM devices comprise eight internal leaves. The DMCU controls the leaf selects within 1 Gbit and 2 Gbit DDR2 SDRAM by toggling **BA[0]**, **BA[1]** and **BA[2]**.

The two DDR SDRAM chip enables (**CS[1:0]#**) support a DDR SDRAM memory subsystem consisting of up to two banks. The base address for the two contiguous banks are programmed in the DDR SDRAM Base Register (SDBR) and DDR SDRAM Upper Base Address Register (SDUBR) and must be aligned to the size of supported memory. For example, when the amount of DDR memory supported is 4 GBytes, the base address registers boundary must be programmed to align on a 4 Gbyte boundary. Similarly, when the amount of DDR memory supported is 256 MBytes, the base address registers boundary must be programmed to align on a 256 Mbyte boundary. The bank size of the DDR SDRAM bank is programmed in the DDR SDRAM Bank Size registers (SBSR). Bank 0 is also configurable into multiple regions. By default bank 0 is a single 64-bit region. For higher core data processing performance, a 32-bit region can be defined within Bank 0 with the DDR SDRAM 32-bit Region Size Register (S32SR).

Note: The Base Address registers (SDBR and SDUBR) must be programmed to align on boundaries that are based on the total size of DDR memory supported. For example, when only 4Gbytes of DDR memory is supported, then the base address registers must be aligned on a 4-Gbyte boundary.

Note: The DDR SDRAM Bank Size register (SBSR) provides the size of one DDR memory bank. All banks must be of the same size and type. Otherwise, the SBSR is programmed with the smallest bank size.

Table 359. DDR SDRAM Address Register Summary

DDR SDRAM Address Register	Definition
DDR SDRAM Base Register (SDBR)	Provides lower part of the base address for DDR SDRAM memory space. Smallest address alignment supported is 128 Mbyte boundary. This register is used in conjunction with SDUBR to provide the entire base address.
DDR SDRAM Upper Base Register (SDUBR)	Upper part of the base address for DDR SDRAM memory space. This register is used in conjunction with SDBR to provide the entire base address.
DDR SDRAM Size Register (SBSR)	Provides size of a single bank. Smallest size is 128 Mbytes.
DDR SDRAM 32-Bit Size Register (S32SR)	The size for the memory space to operate as 32-bit memory region (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: DDR SDRAM memory space must be aligned to a 128 Mbyte boundary and must **never** cross a 4 Gbyte boundary.

Figure 80 illustrates the 81341 and 81342 DDR SDRAM 64-bit Memory Address Map, and Figure 81 illustrates the Physical map of the 64-bit DDR SDRAM. The size of each region is configured separately. The 32-bit region is only applicable when the 81341 and 81342 is connected to 64-bit DDR SDRAM memory. With 32-bit DDR SDRAM attached to 81341 and 81342, all DDR SDRAM memory space behaves as 32-bit DDR SDRAM and the value in S32SR is ignored.

When the 32-bit region defined by S32SR is greater than zero, bank 0's address space is split into three regions as illustrated in Figure 80. The lowest addressable region operates as a 32-bit region of size defined in the S32SR. A second region, contiguous with the 32-bit region, and equal in size, is defined as invalid for the DDR SDRAM memory space. This invalid region reflects the other half of the 64-bit DDR SDRAM which is not used in the 32-bit region. Transactions which address this region results in an error and interrupt to the Intel XScale® microarchitecture. The third region is contiguous with the second and is the remainder of bank 0 address space, and is a 64-bit region.

Figure 80. DDR SDRAM 64-bit Memory Address Map

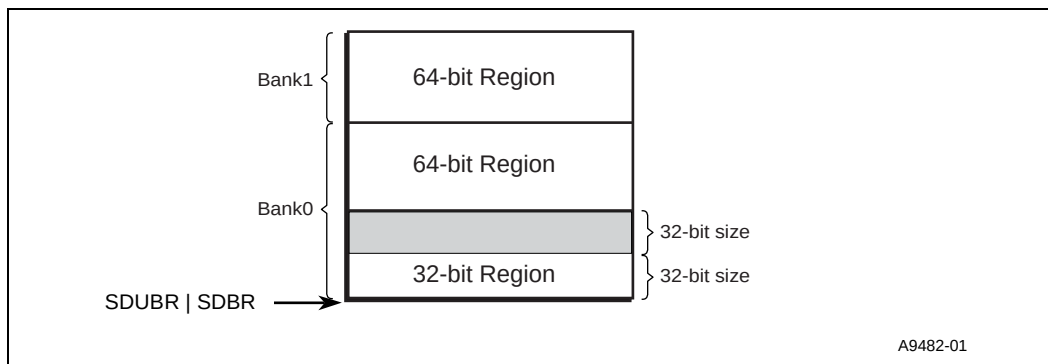
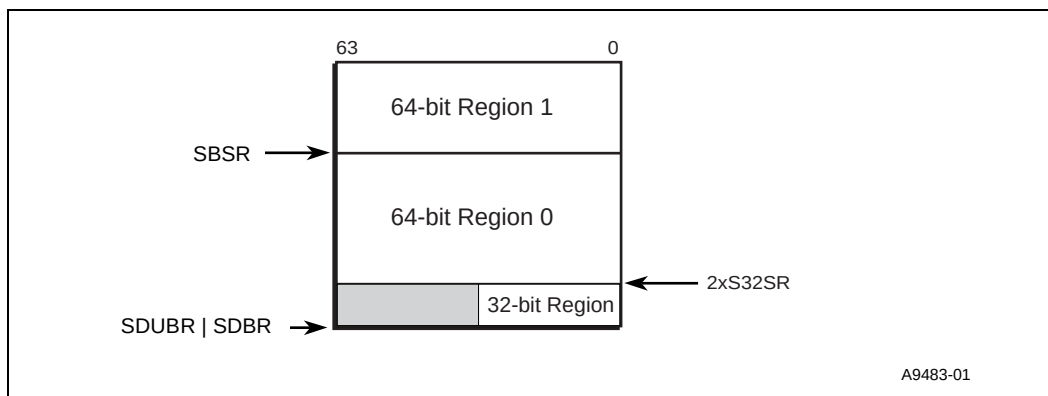


Figure 81. DDR SDRAM 64-bit Physical Map



The base address registers provide the starting address of the DDR memory space, and must be aligned to the size of supported DDR memory. The smallest DDR memory bank size supported is 256 MBytes. In this case, the base address registers (SDBR and SDUBR) define the upper eight address bits of the starting address of the DDR SDRAM memory space. The largest DDR memory supported is 4 GBytes. In this case, the base address registers (SDBR and SDUBR) define the upper four address bits of the starting address of the DDR SDRAM memory space.



To support 4 GBytes of SDRAM memory, the entire memory has to be mapped in an aligned 4-GByte address space of the 64-GByte address space. Note that some of the 64-GByte of the address space are reserved. Since the lowest 4-GByte (36-bit internal address bits [35:32] = 0H) of the address space also contains other peripheral devices, the 4-GByte SDRAM memory must be mapped in one of the upper 4-GByte of the address space in order to be able to utilize the entire 4-GByte memory.

However, the Intel XScale® microarchitecture Operating System (OS) must have access to some SDRAM memory space in the lowest 4-GByte of the address space. This is because after reset the Intel XScale® microarchitecture only accesses the lowest 4-GByte of the address space. To support the Intel XScale® microarchitecture requirements, two sets of base address/size fields are provided by the DMCU to allow forming two separate memory windows.

- A Primary Memory Window is setup to map the entire 4-GByte of SDRAM memory in an open upper 4-GByte of the address space.
- A Secondary Memory Window is setup to map part of the SDRAM memory in the lowest 4-GByte of the address space.

Note that the secondary memory window may be smaller than the primary memory window, and that the secondary memory window must never be larger than the total SDRAM size defined by the SBSR, or 2 GByte, whichever is smaller. With the secondary memory window, the Intel XScale® microarchitecture runs an Operating System (OS) to access part of the SDRAM memory along with other peripheral devices all residing in the lowest 4-GByte of the address space.

The primary memory window is used to access the entire 4-GByte of the SDRAM memory by other internal bus masters like the Address Translation Units and the DMA Engines.

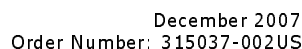
Note: The same SDRAM memory space allocated and used via the Secondary Window by the Operating System must not be used by other internal bus masters via the Primary Memory Window.

The primary memory window is setup using the base address registers: [Table 377, “SDRAM Base Register — SDBR” on page 632](#) and [Table 378, “SDRAM Upper Base Register — SDUBR” on page 633](#) the size field (bits [31:27]) and the number of bank field (bit[2]) of [Table 380, “SDRAM Bank Size Register — SBSR” on page 635](#).

Warning: Setting up a Secondary Memory Window is optional when the amount of SDRAM memory supported is mapped in the lowest 4-GByte of the address space along with the peripheral devices. In this case, only the Primary Memory Window is setup, and the Secondary Memory Window is disabled (SBSR[21:16] = 0H). By default both memory windows are disabled.

Warning: When supporting both the Primary and Secondary Memory Windows, the Programmer must always initialize and setup the DMCU with respect to the Primary Memory Window first, as if the Secondary Memory Window does not exist. Only after setting up the Primary Memory Window, the Programmer can setup and enable the Secondary Memory Window as explained below. This is because the Secondary Memory Window is used in conjunction with the Primary Memory Window.

Figure 82. Secondary DDR SDRAM Window Memory Map





When the secondary memory window address range is defined outside the defined primary memory window, access made to the SDRAM memory via to the secondary memory window may not be able to access the SDRAM memory array, or may access unintended parts of the SDRAM memory array. If the secondary memory window address range is defined outside the primary memory window, transactions made to the secondary memory window will be claimed, but the DMCU discards the transactions and will post an Address Range Error, bit 3 of the [Table 393, "DDR Memory Controller Interrupt Status Register — DMCISR"](#) on page 647.

Since the DMCU maintains memory coherency as described in section [Section 7.3.2.7, "DMCU Port Coherency"](#) on page 571, the upper four bits [35:32] of the secondary memory window address are translated by the value provided in the ["SDRAM Upper Base Register — SDUBR"](#) on page 633. The translated address, which is now a primary memory window address, is then used by the coherency logic. In addition, since the DMCU translates the secondary memory window address to a primary memory window address, errors are logged using the primary memory window address.

Note: Accessing the SDRAM memory via the Secondary Memory Window must only be performed by the Intel XScale® microarchitecture. Other Internal Bus Masters like the Address Translation Units and the DMA Engines must access the SDRAM memory only via the Primary Memory Window.



Table 360 shows correct programming values for DDR SDRAM Bank Size and Table 361 shows correct values for number of DDR SDRAM Banks supported. Bank size and number of banks values are used to calculate the total size of DDR memory supported. Calculated value also provides the boundary the Base Address Registers is programmed with. Base Address registers must be aligned to the total size of DDR memory supported. Table 362 shows the correct programming values for the secondary DDR SDRAM Bank Size.

Table 360. Programming Values for the DDR SDRAM Bank Size Register

Bank Size	SDSR[31:27]	Bank Size	SDSR[31:27]
Empty	00000 ₂	512 MB	00100 ₂
128 MB ¹	00001 ₂	1 GB	01000 ₂
256 MB	00010 ₂	2 GB	10000 ₂

1. 128 MB bank size is capable only when using 32-bit wide memory.

Table 361. Programming Values for the DDR SDRAM Bank Number

Number of Banks	SBSR[2]
1	1 ₂
2	0 ₂

Table 362. Programming Values for the Secondary DDR SDRAM Window Size

Bank Size	SBSR[21:16]
Empty	00 0000 ₂
64 MB	00 0001 ₂
128 MB	00 0010 ₂
256 MB	00 0100 ₂
512 MB	00 1000 ₂
1 GB	01 0000 ₂
2 GB	10 0000 ₂

Note: For Table 362, the secondary memory window must never be larger than the total SDRAM size defined by the SBSR, or 2 GByte, whichever is smaller.

Table 363 shows correct programming values for 32-bit DDR SDRAM Size Register. The maximum size of the 32-bit region is one-half of Bank-0 size (all banks are same size) or 512 MBytes, whichever is smaller.

Table 363. Programming Values for the DDR SDRAM 32-bit Size Register (S32SR[29:20])

32-bit Region Size	S32SR[29:20]	32-bit Region Size	S32SR[29:20]
Empty	000H	32M	020H
1M	001H	64M	040H
2M	002H	128M	080H
4M	004H	256M	100H
8M	008H	512M	200H
16M	010H	reserved	all other values



Example 9. DDR Registers Programming Example 1

The user wants to program the DDR registers to support two banks and each DDR bank is 256 MBytes, yielding in a total memory of 512 MBytes (2*256MB). The user wants the DDR memory space to start at C E000 0000H. There is no 32-bit memory region and no secondary window. The memory space summary is:

The registers would be programmed as follows:

Bank Size = 256 MB, SBSR[31:27] = 00010₂
 Number of Banks = 2, SBSR[2] = 0₂
 S32SR[29:20] = 0H, S32SR = 00000000H
 SDUBR = 0000 000CH, SDUBR[3:0] = 1100₂
 SDBR = E000 0000H, SDBR[31:28] = 1110₂
 SSDBR = Not Applicable, as Secondary Bank Size is Empty.
 Secondary Bank Size = Empty, SBSR[21:16] = 000000₂

Example 10. Address Register Programming Example 2

The user wants to program the DDR registers to support two banks and each DDR bank is 512 MBytes, yielding in a total memory of 1 GBytes (2*512 MB). The user wants the DDR memory space to start at A C000 0000H. There is no 32-bit memory region and no secondary window. The memory space summary is:

The registers are programmed as follows:

Bank Size = 512 MB, SBSR[31:27] = 00100₂
 Number of Banks = 2, SBSR[2] = 0₂
 S32SR[29:20] = 0H, S32SR = 00000000H
 SDUBR = 0000 000AH, SDUBR[3:0] = 1010₂
 SDBR = C000 0000H, SDBR[31:28] = 1100₂
 SSDBR = Not Applicable, as Secondary Bank Size is Empty.
 Secondary Bank Size = Empty, SBSR[21:16] = 000000₂

Example 11. DDR Registers Programming Example 3

The user wants to program the DDR registers to support one bank and the DDR bank is 1 GBytes, yielding in a total memory of 1 GBytes (1*1GB). The user wants the DDR memory space to start at C 4000 0000H. There is no 32-bit memory region. The memory space summary is:

The registers would be programmed as follows:

Bank Size = 1 GB, SBSR[31:27] = 01000₂
 Number of Banks = 1, SBSR[2] = 1₂
 S32SR[29:20] = 0H, S32SR = 00000000H
 SDUBR = 0000 000CH, SDUBR[3:0] = 1100₂
 SDBR = 4000 0000H, SDBR[31:28] = 0100₂
 SSDBR = Not Applicable, as Secondary Bank Size is Empty.
 Secondary Bank Size = Empty, SBSR[21:16] = 000000₂



Example 12. DDR Registers Programming Example 4

The user wants to program the DDR registers to support two banks and each DDR bank is 256 MBytes, yielding in a total memory of 512 G Byte (2*256 MB). The user wants the DDR memory space to start at B E000 0000H. The 32-bit memory region is 128 Mbytes (Bank 0 is entirely 32-bit region since half of the bank space is lost). The memory space summary is:

The registers would be programmed as follows:

Bank Size = 256 MB, SBSR[31:27] = 00010₂
Number of Banks = 2, SBSR[2] = 0₂
S32SR[29:20] = 080H, S32SR = 0800 0000H
SDUBR = 0000 000BH, SDUBR[3:0] = 1011₂
SDBR = E000 0000H, SDBR[31:28] = 1110₂,
SSDBR = Not Applicable, as Secondary Bank Size is Empty.
Secondary Bank Size = Empty, SBSR[21:16] = 000000₂

Note: In the above example, although 32-bit region is enabled, the entire DDR memory space is still configured as supporting a 64-bit wide data bus, that is SDCR0[1] = 0₂.

Example 13. DDR Registers Programming Example 5

The user wants to program the DDR registers to support 1 bank of 32-bit memory only. The smallest DDR2 size that can be supported is 128 M Bytes, as dictated by the DDR2 technology. The user wants the DDR memory space to start at B F800 0000H. The memory space summary is:

The registers would be programmed as follows:

Bank Size = 128 MB, SBSR[31:27] = 00001₂
Number of Banks = 1, SBSR[2] = 1₂
S32SR[29:20] = 000H, S32SR = 0000 0000H
SDCR0[1] = 1₂, Enable 32-bit data bus width
SDUBR = 0000 000BH, SDUBR[3:0] = 1011₂
SDBR = F800 0000H, SDBR[31:27] = 11111₂
SSDBR = Not Applicable, as Secondary Bank Size is Empty.
Secondary Bank Size = Empty, SBSR[21:16] = 000000₂

Example 14. Address Register Programming Example 6

The user wants to program the DDR registers to support two banks and each DDR bank is 512 MBytes, yielding in a total memory of 1 GBytes (2*512 MB). The user wants the DDR memory space to start at A C000 0000H. The user wants a 256-MByte secondary window starting at 0 C00 0000H. Note that the secondary memory window overlaps the primary memory window and starts at the bottom of the primary memory window in this example. There are no 32-bit memory region. The memory space summary is:

The registers would be programmed as follows:

Bank Size = 512 MB, SBSR[31:27] = 00100₂
Number of Banks = 2, SBSR[2] = 0₂
S32SR[29:20] = 0H, S32SR = 00000000H
SDUBR = 0000 000AH, SDUBR[3:0] = 1010₂
SDBR = C000 0000H, SDBR[31:28] = 1100₂
SSDBR = C000 0000H
Secondary Bank Size = 256 MB, SBSR[21:16] = 000100₂

**Example 15. Address Register Programming Example 7**

The user wants to program the DDR registers to support two banks and each DDR bank is 2 GBytes, yielding in a total memory of 4 GBytes (2*2 GB). The user wants the DDR memory space to start at A 0000 0000H. The user wants a 2-GByte secondary window starting at 0 0000 0000H. Note that the secondary memory window overlaps the primary memory window and starts at the bottom of the primary memory window in this example. There are no 32-bit memory region. The memory space summary is:

The registers would be programmed as follows:

Bank Size = 2 GB, SBSR[31:27] = 10000₂
Number of Banks = 2, SBSR[2] = 0₂
S32SR[29:20] = 0H, S32SR = 00000000H
SDUBR = 0000 000AH, SDUBR[3:0] = 1010₂
SDBR = 0000 0000H, SDBR[31:28] = 0000₂
SSDBR = 0000 0000H
Secondary Bank Size = 2 GB, SBSR[21:16] = 100000₂



7.3.3.4 32-bit Data Bus Width

Using 512 Mbit (with x16 devices) SDRAMs, a 64-bit data bus yields a minimum memory size of 256 MBytes. To address cost-sensitive applications requiring less than 256 MBytes of local memory, the DMCU supports a 32-bit data bus. While 32-bit mode decreases the minimum memory size to 128 MBytes, the bus throughput is also reduced to half the throughput of 64-bit mode. For example, the throughput for DDR2 400 in 64-bit mode and 32-bit mode is 3200 MBytes/s and 1600 MBytes/s respectively.

The DMCU does not support switching between 32-bit data bus width and 64-bit data bus width

Figure 83. 64-bit to 32-bit Addressing

SDRAM Column Address on MA[13:0]			Address for 32-bit Data		
Example assumes that the 32-bit address in question has the same row address independent of memory bus width.			9	Data 9	24H
			8	Data 8	20H
			7	Data 7	1CH
			6	Data 6	18H
			5	Data 5	14H
			4	Data 4	10H
			3	Data 3	0CH
			2	Data 2	08H
			1	Data 1	04H
			0	Data 0	00H

Address for 64-bit Data	
20H	Data 9 Data 8
18H	Data 7 Data 6
10H	Data 5 Data 4
08H	Data 3 Data 2
00H	Data 1 Data 0

B6257-01

B6257-01

The data bus width is selected by bit 2 of the SDCR0 (see [Section 7.8.2, “SDRAM Control Register 0 — SDCR0” on page 628](#)). The default is 64-bit bus width.

Reducing the data bus width by half also reduces the page size by half. Therefore, the page size is 4 Kbytes for 32-bit data bus versus 8 Kbytes for 64-bit data bus width.



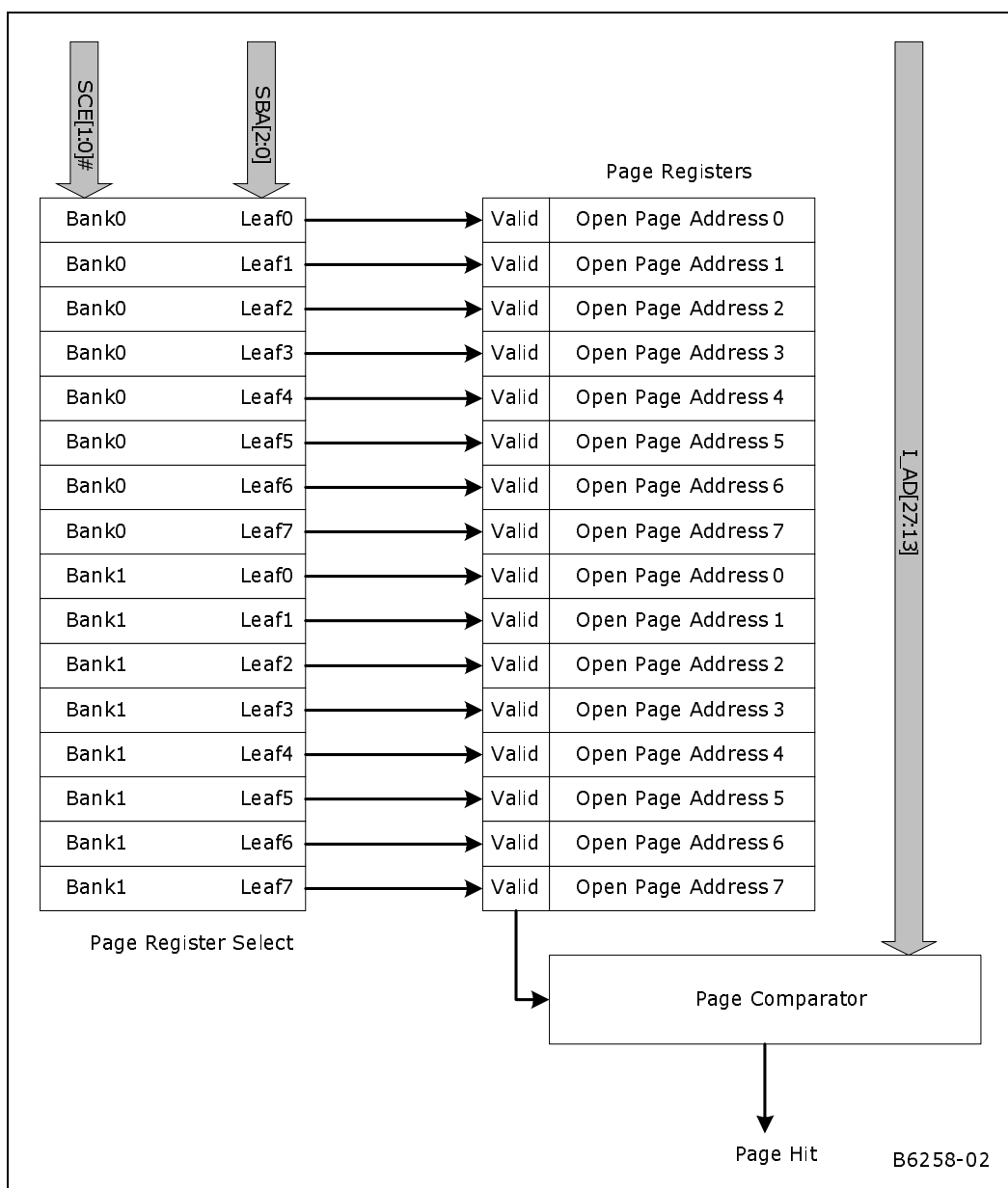
7.3.3.5 Page Hit/Miss Determination

The DMCU address translation assumes a 8 Kbyte page. For 32-bit mode, the DMCU address translation assumes a 4 Kbyte page. For 512 Mbit DDR2 devices, the DMCU keeps four pages per bank (8 maximum) open simultaneously. And for 1 Gbit and 2 Gbit DDR2 devices, the DMCU keeps eight pages per bank (16 maximum) open simultaneously.

For 512 Mbit DDR2 devices, the DMCU keeps only one page each of Bank0 (Leaf [0:3]) and Bank1 (Leaf [0:3]) open simultaneously.

For 1 Gbit and 2 Gbit DDR2 devices, the DMCU keeps only one page each of Bank0 (Leaf [0:7]) and Bank1 (Leaf [0:7]) open simultaneously. See [Figure 85](#) for an example organization using 1 Gbit and 2 Gbit devices.

Figure 84. Page Hit/Miss Logic for 1Gbit Mode





The DMCU logic determines the hit/miss status for reads and writes. For a new DDR SDRAM transaction, the DMCU compares the address of the current transaction with the address stored in the appropriate page address register. Assuming 512 Mbit DDR2 SDRAM devices and two banks, there are eight pages kept open simultaneously. The DDR SDRAM chip enables (**CS[1:0]#**) and leaf selects (**BA[2:0]**) determine which page address to compare. BA2 is not available on 512 Mbit DDR2 SDRAM devices. For 1 Gbit/2 Gbit DDR2 SDRAM devices and two banks, there are sixteen pages kept open simultaneously. The DDR SDRAM chip enables (**CS[1:0]#**) and leaf selects (**BA[2:0]**) determine which page address to compare.

When the current transaction misses the open page selected then the DMCU closes the open page pointed to by **CS[1:0]#** and **BA[2:0]** by issuing a **precharge** command. The DMCU opens the current page with a **row-activate** command and the transaction completes with a **read** or **write** command. When the DMCU opens the current page, **AD[35:13]** from the corresponding memory transaction queue is stored in the page address register pointed to by **CS[1:0]#** and **BA[2:0]** so it may be compared for future transactions.

For a page hit, the DMCU does not need to open the page (assert **RAS#**) and avoids the RAS-to-CAS delay achieving greater performance. For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.

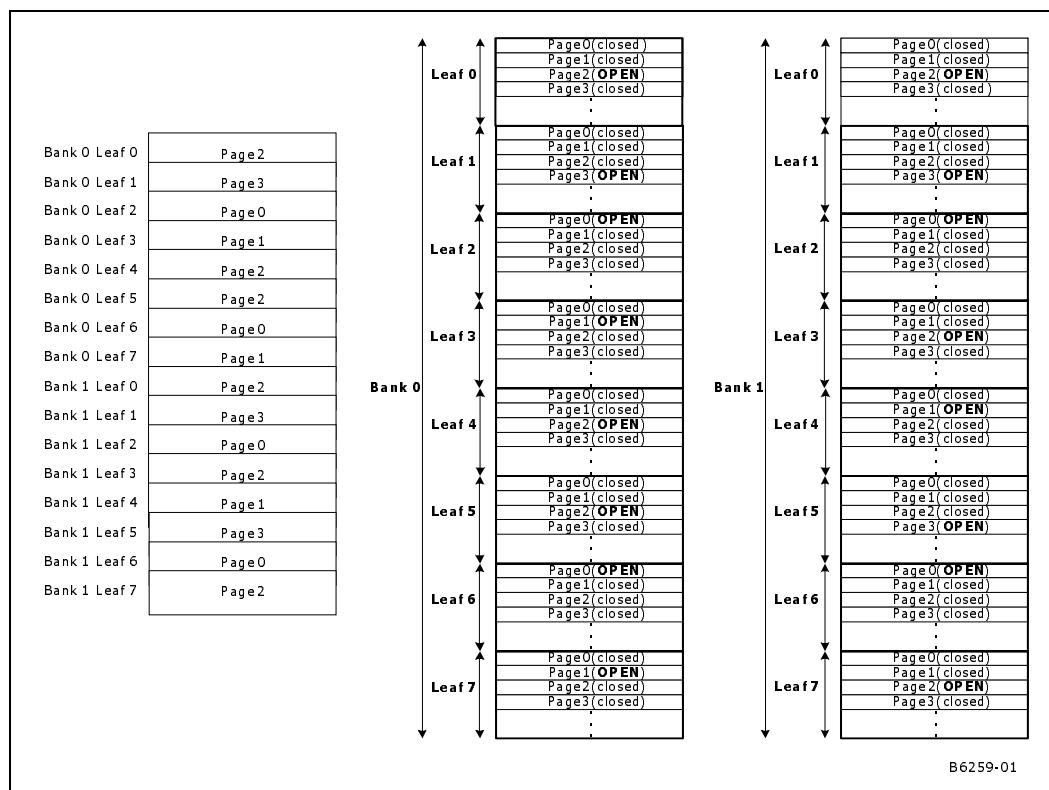
When the current transaction hits the open page, then the page is already active and the **read** or **write** command may be issued without a **row-activate** command. When the next transaction is the same command type as the current transaction, and also a page hit, the DMCU does not need to issue the command again, but simply drive column address for an open page.

When the refresh timer expires and the DMCU issues an **auto-refresh** command, all pages are closed and the valid bits are cleared.

Figure 85 illustrates how the logical memory image is partitioned with respect to open and closed pages. When the above image represents a 128 Mbyte DDR 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 block sizes depend on the memory sizes implemented in the DDR SDRAM memory subsystem. The page size is always 8 Kbytes for a 64-bit wide memory data bus, and 4 Kbytes for a 32-bit wide memory data bus. The programmer can optimize DDR SDRAM transactions by partitioning code and data across the leaf boundaries to maximize the number of page hits.

Figure 85. Logical Memory Image of a DDR SDRAM Memory Subsystem





7.3.3.6 DDR SDRAM Commands

The DMCU issues specific commands to the DDR SDRAM devices by encoding them on the **CS[1:0]#**, **RAS#**, **CAS#**, and **WE#** inputs. Table 364 lists all of the DDR SDRAM commands understood by DDR SDRAM devices. The DMCU supports a subset of these commands.

Table 364. DDR SDRAM Commands

Command ^{a, b}	Conditions					Comments
	SCE#	RAS#	CAS#	SWE#	Other	
NOP	0	1	1	1		No Operation
Load Mode	0	0	0	0	BA[1:0] = Sel ^c	Load the (Extended) Mode Register from MA[13:0]
Row Activate	0	0	1	1	BA[1:0] = Leaf	Activate a row specified on MA[13:0]
Read	0	1	0	1	BA[1:0] = Leaf MA[10] = 0	Column burst read Column address on MA[13:0]
Read w/ Auto-Precharge	0	1	0	1	BA[1:0] = Leaf MA[10] = 1	Column burst read with row precharge at the end of the transfer
Write	0	1	0	0	BA[1:0] = Leaf MA[10] = 0	Column burst write Column address on MA[13:0]
Write w/ Auto-Precharge	0	1	0	0	BA[1:0] = Leaf MA[10] = 1	Column burst write with row precharge at the end of the transfer
Precharge	0	0	1	0	BA[1:0] = Leaf MA[10] = 0	Precharge a single leaf
Precharge All	0	0	1	0	MA[10] = 1	Precharge all leaves
Auto-Refresh	0	0	0	1		Refresh both banks from on-chip refresh counter
Self-Refresh	0	0	0	1	CKE = 0	Refresh autonomously while CKE = 0
Power Down	X	X	X	X	CKE = 0	Power down when both banks precharged when CKE = 0
Stop	0	1	1	0		Interrupt a read or write burst.

a. Shaded boxes indicate commands not supported by 81341 and 81342. They are included for completeness.

b. 512 Mbit DDR2 technologies uses **BA[1:0]**, whereas 1 Gbit/2 Gbit DDR2 technologies use **BA[2:0]**.

c. Selects which mode register is programmed. 00₂ selects Mode Register, 01₂ selects Extended Mode Register, 10₂ selects Extended Mode Register 2, and 11₂ selects Extended Mode Register 3.

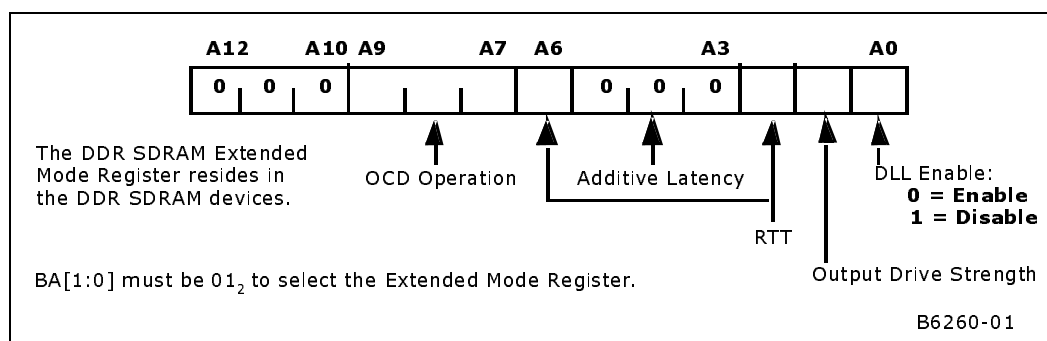
DDR SDRAM commands are synchronous to the clock so the DMCU sets up the above conditions prior to the **M_CK[2:0]** rising edge.

7.3.3.7 DDR SDRAM Initialization

Since DDR SDRAM devices contain a controller within the device, the DMCU must initialize them specifically. Upon the deassertion of internal bus reset, software initializes the DDR SDRAM devices with the sequence illustrated with Figure 88. However, before executing the DDR SDRAM initialization sequence provided below, software must read the SPD (Serial Presence-Detect) data and then program the DMCU parameters as explained in Section 7.3.3.8, “DDR SDRAM Mode Programming” on page 601. The DMCU does **not** interact properly with the DDR SDRAM until these parameters have been programmed.

1. The DMCU applies the clock (**M_CK[2:0]**) at power up along with system power (clock frequency unknown).
2. The DMCU must stabilize **M_CK[2:0]** within 100 μ s after power stabilizes.
3. The DMCU holds all the control inputs inactive (**RAS#**, **CAS#**, **WE#**, **SCE[1:0]# = 1**), places all data outputs and strobes in the High-Z state (**DQS[8:0]**, **DQS[8:0]#**, **DQ[71:0]**), and deasserts **CKE[1:0]** for a minimum of 200 μ s after supply voltage reaches the desired level. Asserting **P_RST#** achieves this state.
4. Software disables the refresh counter by setting the RFR to zero.
5. Software issues one **NOP** cycle after the 200 μ s device deselect. A **NOP** is accomplished by setting the SDIR to 0000_0070H. The DMCU asserts **CKE[1:0]** with the NOP.
6. Software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0002_0020H.
7. Software issues an **extended-mode-register-set 2 (EMRS(2))** command by writing 0100_0000H to the SDIR.
8. Software issues an **extended-mode-register-set 3 (EMRS(3))** command by writing 0180_0000H to the SDIR.
9. Software issues an **extended-mode-register-set 1 (EMRS(1))** command to enable DLL by writing 0080_0000H to SDIR. DMCU supports the following DDR and DDR2 SDRAM mode parameters:
 - a. DLL = Enable/Disable
 - b. Off-Chip Driver (OCD) Impedance Adjustment — applies to DDR2 SDRAM only.
 - c. Additive Latency (AL) is always zero for 81341 and 81342.

Figure 86. Supported DDR2 SDRAM Extended Mode Register Settings





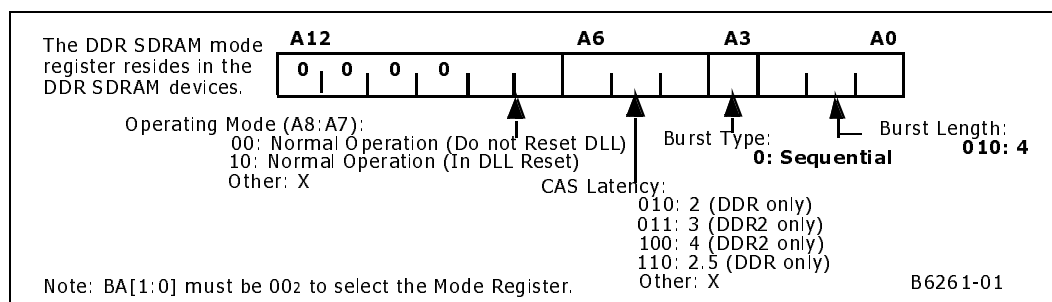
10. After waiting T_{mrd} cycles, software issues a **mode-register-set** command by writing one of the following options to SDIR to program DDR SDRAM parameters and to **Reset** DLL:

- 0002_9900H for CAS Latency = 3 and Write Recovery = 3, or
- 0003_9900H for CAS Latency = 3 and Write Recovery = 4, or
- 0002_A100H for CAS Latency = 4 and Write Recovery = 3, or
- 0003_A100H for CAS Latency = 4 and Write Recovery = 4, or
- 0002_A900H for CAS Latency = 5 and Write Recovery = 3, or
- 0003_A900H for CAS Latency = 5 and Write Recovery = 4

The DMCU supports the following DDR SDRAM mode parameters:

- a. CAS Latency (tCAS) = 3, 4, or 5 for DDR2 based on the programmed setting in “[SDRAM Control Register 0 — SDCR0](#)”. Note, when ODT is enabled CAS Latency must be => 4.
- b. Burst Type = Sequential
- c. Burst Length (BL) = four

Figure 87. Supported DDR SDRAM Mode Register Settings



11. After waiting T_{mrd} cycles, software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0002_0020H.
12. After waiting T_{rp} cycles, software provides two **auto-refresh** cycles, accomplished by setting SDIR to 0000_0010H, then ensure at least T_{rfc} cycles between each **auto-refresh** command.
13. After the second **auto-refresh** cycle, software must wait T_{rfc} cycles. Then, software issues a **mode-register-set** command by writing one of the following options to the SDIR to program DDR SDRAM parameters **without** resetting DLL:
 - 0002_1900H for CAS Latency = 3 and Write Recovery = 3, or
 - 0003_1900H for CAS Latency = 3 and Write Recovery = 4, or
 - 0002_2100H for CAS Latency = 4 and Write Recovery = 3, or
 - 0003_2100H for CAS Latency = 4 and Write Recovery = 4, or
 - 0002_2900H for CAS Latency = 5 and Write Recovery = 3, or
 - 0003_2900H for CAS Latency = 5 and Write Recovery = 4
14. DMCU may issue a **row-activate** command T_{mrd} cycles after **mode-register-set** command.
15. Software re-enables the refresh counter by setting the RFR to the required value.



Table 365. SDIR Encoding Examples (Sheet 1 of 3)

SDIR Bit	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	SDIR Encoded Value				
	C0C1#	C0C1#	RSV	RSV	RSV	BA2	BA1	BA0	RSV	RSV	RSV	MA12	MA11	MA10	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0	RAS#	CAS#	WE#	DT3	DT2	DT1	DT0		
Command																																	
DDR2 Commands																																	
NOP	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X	X	X	0000 0070H	
Precharge All	0	0	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	0	1	0	X	X	X	X	0002 0020H	
Auto Refresh	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1	X	X	X	X	0000 0010H	
MRS Commands ^a																																	
MRS ^b (Parameter Fields)	L	L	X	X	X	X	MR ^c		X	X	X	P ^d		WR ^d		D ^e	L ^e	T ^e	CAS# Latency	B ^f	Burst Length		L	L	L	X	X	X	X				
MRS Command (tWR = 3, Reset DLL, Set CAS Latency = 3, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0	X	X	X	X	0002 9900H
MRS Command (tWR = 3, Do NOT reset DLL, Set CAS Latency = 3, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	X	X	X	X	0002 1900H
MRS Command (tWR = 4, Reset DLL, Set CAS Latency = 3, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	X	X	X	X	0003 9900H
MRS Command (tWR = 4, Do NOT reset DLL, Set CAS Latency = 3, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	1	0	0	0	1	1	0	0	1	0	0	0	0	X	X	X	X	0003 1900H
MRS Command (tWR = 3, Reset DLL, Set CAS Latency = 4, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	X	X	X	X	0002 A100H
MRS Command (tWR = 3, Do NOT reset DLL, Set CAS Latency = 4, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	X	X	X	X	0002 2100H
MRS Command (tWR = 4, Reset DLL, Set CAS Latency = 4, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	1	1	0	1	0	0	0	1	0	0	0	0	0	X	X	X	X	0003 A100H
MRS Command (tWR = 4, Do NOT reset DLL, Set CAS Latency = 4, Set Burst Type = Sequential, Set Burst Length = 4)	0	0	X	X	X	X	0	0	0	X	X	X	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	X	X	X	X	0003 2100H



Table 365. SDIR Encoding Examples (Sheet 2 of 3)

[illegible]



EMRS Command (Output = enabled, RDQS Enable = Yes, DQS# Enable = enable, OCD Program = exit, RTT = Disabled, Additive latency = 000b, disableDLL)		0	0	X	X	X	X	X	O	O	1	X	X	X	X	O	O	O	O	O	O	O	O	O	O	1	O	O	O	X	X	X	X	0080	0080H		
EMRS Command (Output = enabled, RDQS Enable = Yes, DQS# Enable = enable, OCD Program = exit, RTT = 75 ohm, Additive latency = 000b, disableDLL)		0	0	X	X	X	X	O	O	1	X	X	X	X	O	O	O	O	O	O	O	O	O	O	O	1	O	1	O	O	O	X	X	X	X	0080	0280H
EMRS Command (Output = enabled, RDQS Enable = Yes, DQS# Enable = enable, OCD Program = exit, RTT = 150 ohm, Additive latency = 000b, disableDLL)		0	0	X	X	X	X	O	O	1	X	X	X	X	O	O	O	O	O	O	O	O	O	O	O	1	O	O	O	O	X	X	X	X	0080	2080H	

- MRS Command examples are shown for DDR2 at 400MHz and 533Hz.
- Refer to [Table 367](#) for Mode Register parameter definitions.
- Set to 000b for MR Command.
- tWR is 3 and 4 for 400MHz and 533MHz respectively for DDR2.
- MRS Command examples are shown for DDR2 at 400MHz and 533Hz.
- Refer to [Table 367](#) for Mode Register parameter definitions.
- Set to 000b for MR Command.
- tWR is 3 and 4 for 400MHz and 533MHz respectively for DDR2.
- Refer to [Table 366](#) for Extended Mode Register parameter definitions.
- Set to 001b for EMR Command.

**Table 366. SDRAM Extended Mode Register (EMRS) Definition**

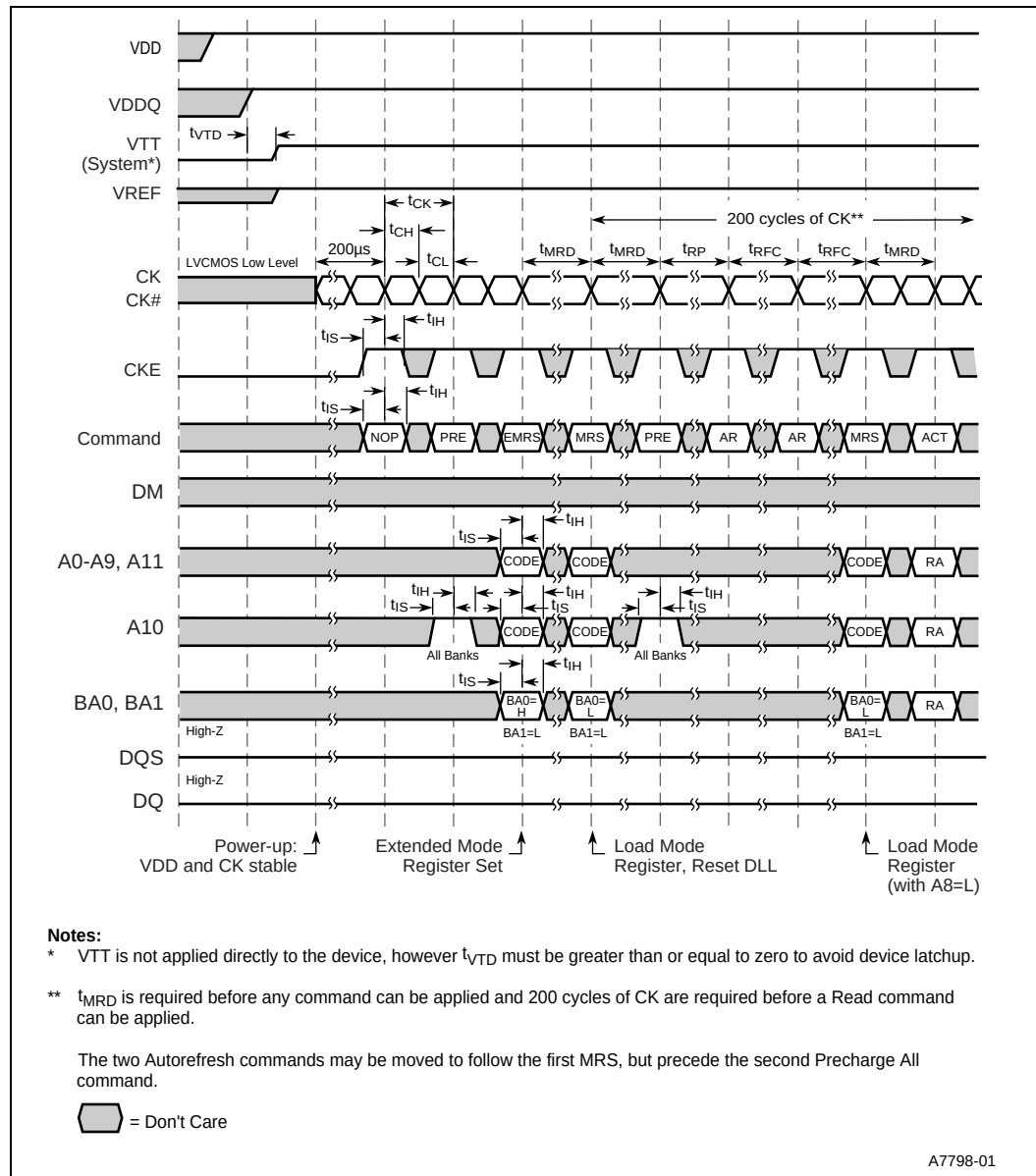
Extended Mode Register Bit	Extended Mode Register Field	Field Definition
14:13	EMRS	MRS Mode (a value of 01b selects Extended Mode Register Select)
12	OUT	Output Buffer Enable/Disable
11	RDQS	RDQS Enable
10	DQS#	DQS# Enable
09:07	OCD Program	OCD Operation
06	RTT	RTT
05:03	Posted CAS#	Posted CAS# Additive latency (AL)
02	RTT	RTT
01	ODS	Out Drive Strength
00	DLL	DLL Enable

Table 367. SDRAM Mode Register (MRS) Definition

Mode Register Bit	Mode Register Field	Field Definition
14:13	MRS	MRS Mode Set (a value of 00b selects Mode Register select)
12	PD	PD Mode (Power-Down Mode)
11:09	WR	Write Recovery
08	DLL	DLL Reset
07	TM	Mode
06:04	CAS# Latency	CAS Latency
03	BT	Burst Type
02:00	BL	Burst Length

The waveform in Figure 88 illustrates the DDR SDRAM initialization sequence.

Figure 88. DDR SDRAM Initialization Sequence (controlled with software)



When the DDR SDRAM subsystem implements ECC (Section 7.3.4, “DDR Error Correction and Detection” on page 607), then initialization software initializes the entire memory array with the 81341 and 81342. It is important that every memory location has a valid ECC byte. The Application DMA includes a memory block fill mode which can be used to fill the memory array with a constant (Section 5.7.8, “Memory Block Fill Operation” on page 505), thereby initializing the associated ECC bytes in the process. A small portion of memory must be initialized by the Intel XScale® microarchitecture software to store the ADMA descriptor. When the memory array is not initialized, the microarchitecture may attempt to read memory locations beyond the specified word(s). In this case, the DMCU reports an ECC error even though software did not specifically request the un-initialized data.



7.3.3.8 DDR SDRAM Mode Programming

The DMCU programs the DDR SDRAM devices through a **mode-register-set** command. During the initialization sequence this command sets the DDR SDRAM mode register (see [Section 7.3.3.7, “DDR SDRAM Initialization” on page 594](#)) by programming the SDIR and SDCR[1:0].

The DDR 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 SDRAM devices being interfaced to 81341 and 81342. Because the parameters that define the time between allowed commands are programmable, this allows flexibility in the type of DDR device that is selected, in addition to de-coupling the hardware to any frequency dependencies. See [Section 7.8.2, “SDRAM Control Register 0 — SDCR0” on page 628](#) and [Section 7.8.3, “SDRAM Control Register 1 — SDCR1” on page 630](#) for equations and values for programming of the MCU timing parameters.

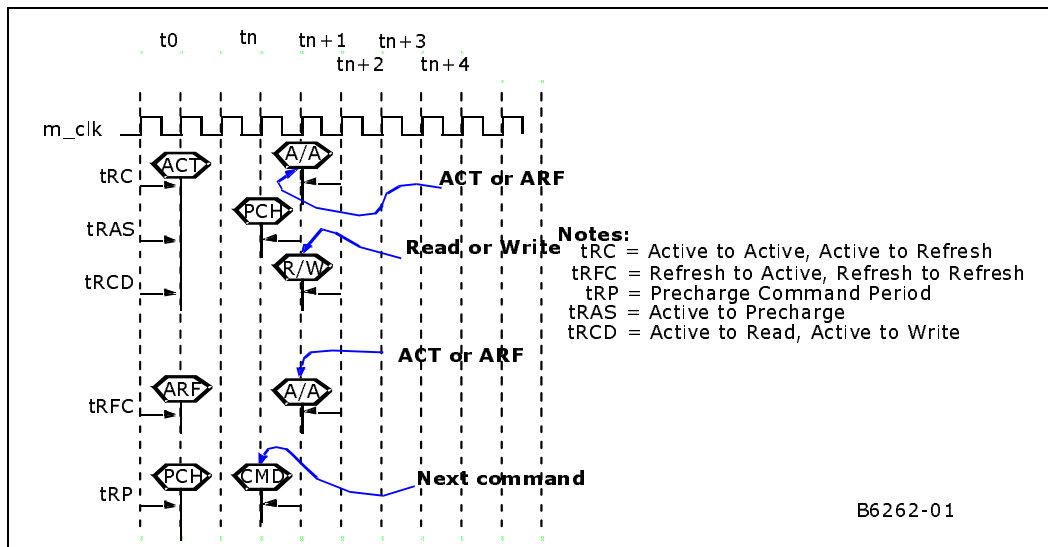
Note: The DMCU_DDRSM does NOT interact properly with the DDR SDRAM until the SDCR[1:0] registers have been programmed.

Table 368. SDCR[1:0] Timing Parameters Summary

Parameter	Source	Use	Parameter	Source	Use
tBL	JEDEC	Equation 31	tRMW	Equation 31	SBSR[11:08]
tCAS	JEDEC	Equation 22 , Equation 31 , Figure 86 , and SDCR0[10:08]	tRTW	Equation 26	SDCR1[22:19]
tEDP	81341 and 81342	SDCR0[18:16]	tWDL	Equation 22	SDCR0[13:12]
tRAS	JEDEC	SDCR0[31:27]	tWR	JEDEC	Equation 25 and SDCR1[11:9]
tRC	JEDEC	SDCR1[08:04]	tWTR	JEDEC	Equation 30
tRCD	JEDEC	SDCR0[22:20]	tWTRD	Equation 30	SDCR1[3:0]
tRFC	JEDEC	SDCR1[17:12]	tWTCMD	Equation 25	SDCR1[26:23]
tRP	JEDEC	SDCR0[26:24]	tRTP	JEDEC	Equation 24
tRTCMD	Equation 24	SDCR1[30:27]			

The timing parameters and protocol for all non-read and non-write commands are derived directly from the JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79, June 200 and JEDEC DDR — II SDRAM Specification, September 2002. Please see the JEDEC specification for the timing parameters specific to the DDR device that is to be implemented in the system.

Figure 89. DMCU Active, Precharge, Refresh Command Timing Diagram



Note: Burst Length (BL) is fixed at four for 81341 and 81342



7.3.3.9 DDR SDRAM Read Cycle

The DMCU performance is optimized for page hits and the DMCUs behavior is different for the hit and miss scenario. For a page hit, the two cycles required for row activation are saved resulting in lower first word read latency.

The DMCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDR SDRAM Control Block to issue the transaction command to the DDR array when the previous transaction is the same type (read or write). In addition, the DDR 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 DDR SDRAM devices.

1. Each of the DMCU inbound memory transaction ports decodes the address to determine when the transaction should be claimed.
 - When the address falls in the DDR SDRAM address range indicated by the SDBR, SDUBR, SBSR, and S32SR the DMCU claims the transaction and latches the transaction in the respective memory transaction queue.
2. Once the DMARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDR SDRAM control block. The DDR 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 7.3.3.5, "Page Hit/Miss Determination" on page 589](#) for the paging algorithm details. When a page hit occurs, steps 3-4 are skipped by the DMCU.

3. The DDR SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row. (Not depicted in [Figure 88](#)).
 - The DDR 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 DDR SDRAM Control Block asserts **RAS#**, de-asserts **WE#**, and drives the row address on **MA[13:0]**.
5. In the following cycle in the case of a page hit or after T_{rdd} cycles in the case of a page miss, the DDR SDRAM Control Block asserts **CAS#**, de-asserts **WE#**, and places the column address on **MA[13:0]**. This initiates the burst read cycle.
6. After the CAS latency expires, the DDR SDRAM device drives data to the DMCU.
7. Upon receipt of the data, the DDR SDRAM Control Block calculates the ECC code from the data and compares it with the ECC returned by the DDR SDRAM array. [Section 7.3.4, "DDR Error Correction and Detection" on page 607](#) explains the ECC algorithm in more detail.
8. Assuming the calculated ECC matches the read ECC, the DDR 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 DMCU continues to return data to the corresponding memory transaction queue based on the byte count of the transaction.



7.3.3.10 DDR SDRAM Write Cycle

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. [Section 7.3.4, “DDR Error Correction and Detection” on page 607](#) explains the ECC algorithm in more detail.

For a page hit, the DMCU does not need to open the page (assert **RAS#**) and avoids the RAS-to-CAS delay achieving greater performance. For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.

1. Each of the DMCU inbound memory transaction ports decodes the address to determine when the transaction should be claimed.
 - When the address falls in the DDR SDRAM address range indicated by the SDBR, SBSR and S32SR the DMCU claims the transaction and latches the transaction in the respective memory transaction queue.
2. Once the DMARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDR SDRAM control block. The DDR 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.

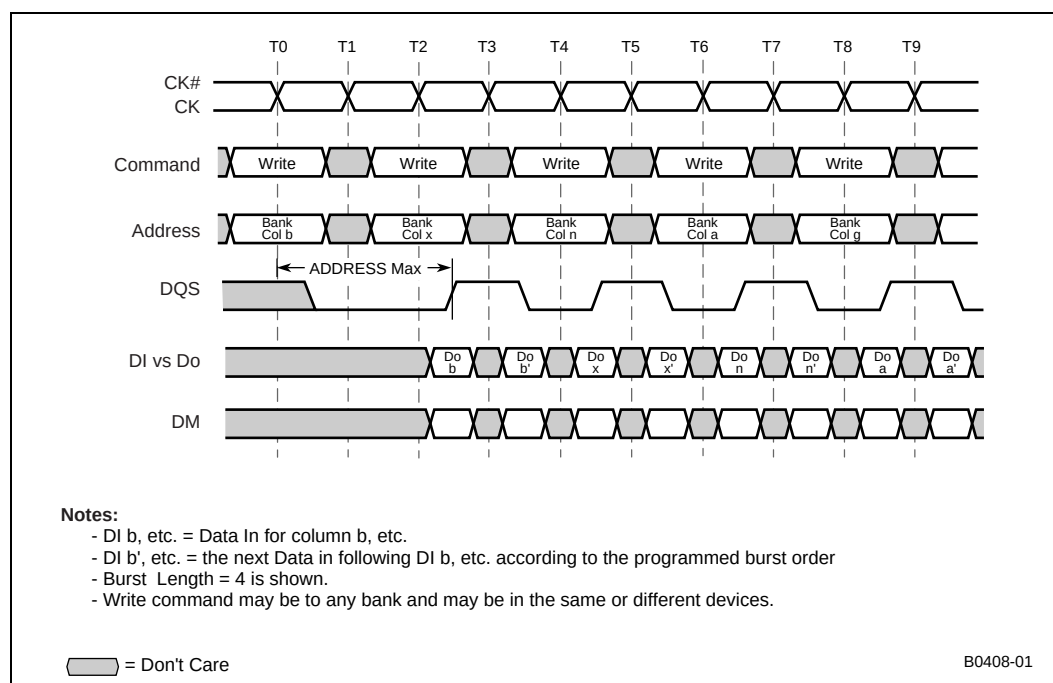
A write that misses the open page encounters a miss penalty because the currently open page needs to be closed before the write can be issued to the new page. Refer to [Section 7.3.3.5, “Page Hit/Miss Determination” on page 589](#) for the paging algorithm details. When a page hit occurs, steps 2-3 are skipped by the DMCU.

3. The DDR SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row.
 - The DDR 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 DDR SDRAM Control Block asserts **RAS#**, de-asserts **WE#**, and drives the row address on **MA[13:0]**.
5. After T_{rzd} cycles in the case of a page miss, the DDR SDRAM Control Block asserts **CAS#**, asserts **WE#**, and places the column address on **MA[13:0]**. This initiates the burst write cycle. The DDR SDRAM Control Block drives the data to be written and its ECC code to the DDR SDRAM devices.
 - The DDR 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 when preemption conditions are met.
 - For each burst issued, the DDR SDRAM Control Block increments the address by four DWORDs (16 bytes) for 32-bit data bus mode, and by four QWORDS (32 bytes) for 64-bit data bus mode.
 - When ECC is enabled, when the data to write is not aligned on an 8 byte boundary (4 byte for 32-bit data bus width or 32-bit region), the DDR SDRAM Control Block performs a read-modify-write of the entire 8 byte aligned quad-word (4 byte aligned double-word for 32-bit data bus width or 32-bit region) and incorporate the new data while regenerating ECC.



The DMCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDR SDRAM Control Block to issue the transaction command to the DDR array when the previous transaction is the same type (read or write). In addition, the DDR 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 DDR SDRAM devices. These optimizations are illustrated in Figure 90 for random write memory transactions.

Figure 90. DDR SDRAM Pipelined Writes





7.3.3.11 DDR SDRAM Refresh Cycle

Since the DDR SDRAM is a dynamic memory, the DMCU issues a refresh cycle periodically. The interval of these refresh cycles is programmable in the RFR register. The DDR SDRAM device generates the refresh address internally. The DMCU initiates two sequential refresh cycles (one per bank) after the DMCUs refresh timer expires and any current transaction is complete. Once the refresh timer expires, the DMCU knows that a refresh cycle is necessary.

- The refresh timer continues to count for the next refresh cycle.
- The DMARB allows the current transaction to complete.
 - When the DDR SDRAM Control Block and the DDR SDRAM array are transferring data, or a NIBPTQ tenure is ongoing, the refresh cycle is queued until the transaction is complete or the NIBPTQ tenure expires.
- The DDR SDRAM Control Block closes all open pages with a **precharge-all** command to all the populated DDR SDRAM banks.
 - The DDR SDRAM Control Block resets the page register valid bits.
- The DDR SDRAM Control Block issues an **auto-refresh** command to DDR SDRAM bank 0.
 - This command affects all internal leaves.
- In the next cycle, the DDR SDRAM Control Block issues an **auto-refresh** command to DDR SDRAM bank 1.
- After T_{rfc} cycles, the DDR SDRAM Control Block can service a new transaction or another refresh cycle.

It is recommended that the RFR ([“Frequency Register — RFR” on page 651](#)) is programmed with the value to achieve 7.8 μ s, though some DDR SDRAM devices may provide for the ability to refresh at a period of 15.6 μ s. The value is based on the frequency of the DDR SDRAM and [Table 369](#) can provides for these two typical values.

Table 369. Typical Refresh Frequency Register Values

DDR Speed	7.8 μ s Value	15.6 μ s Value
400 MHz	618H	C30H
533 MHz	820H	1040H

The longest possible internal bus transaction is writing a 1 Kbyte page where each data cycle results in a read-modify-write due to partial writes (see [Section 7.3.4.2, “DDR ECC Generation for Partial Writes” on page 609](#)). The longest possible NIBPTQ tenure is 16 transactions where each of the transaction are page misses and partial writes. Such periods potentially require queueing two refresh cycles.



7.3.4 DDR Error Correction and Detection

The DMCU is capable of correcting any single bit errors and detecting any double bit errors in the 81341 and 81342 DDR SDRAM memory subsystem. ECC enhances the reliability of a memory subsystem by correcting single bit errors caused by electrical noise or occasional alpha particle hits on the DDR SDRAM devices.

Similar to parity, which simply detects single bit errors, error correction requires an additional 8-bit code word for the 64-bit (32-bit) datum. This means that a memory must have the additional 8-bit error correction code (**CB[7:0]**) per 64-bit (32-bit) datum (**DQ[63:0]**) resulting in a 72-bit (40-bit) wide memory subsystem. During DDR SDRAM read cycles, the DDR SDRAM Control Block detects single bit errors and corrects the data prior to returning the data to the respective memory transaction queue. DDR SDRAM write cycles generate the ECC and sends it with the data to the memories.

In the 32-bit region with 64-bit memory, or with 32-bit wide memory, the 81341 and 81342 zeros 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 results in a 40-bit wide memory since an 8-bit error correction code is still required.

Scrubbing is the process of correcting an error in the memory array. The chance of an unrecoverable multi-bit error increases when the software does not correct a single-bit error in the array. For the 81341 and 81342, scrubbing is handled by software. When error reporting is enabled, the DMCU logs the error type in ELOG0 or ELOG1 and the address in ECAR0 or ECAR1 when an error occurs.

7.3.4.1 DDR ECC Generation

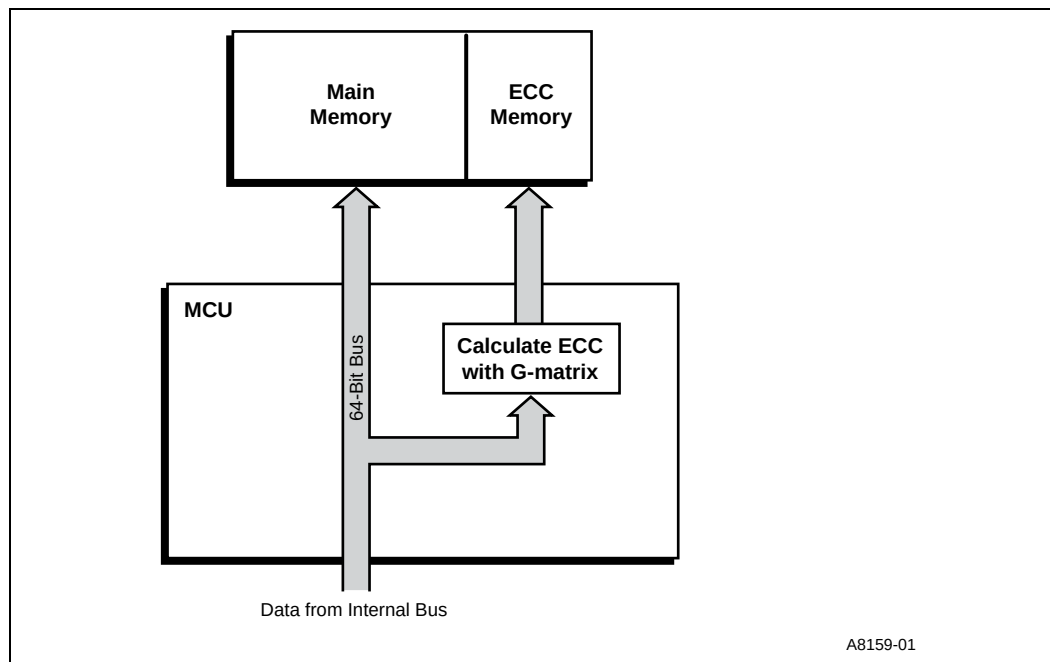
For write operations, the DMCU generates the error correction code which is written along with the data. This section describes the operation of the DDR SDRAM Control Block for ECC generation in a 64-bit wide memory and 64-bit region. The same principles apply for 32-bit wide memory and in the 32-bit region in 64-bit wide memory, however the DMCU generates 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 bits wide
    Generate the ECC_with the G-matrix
    Write the new data and ECC
else {Partial Write}
    Read entire 64-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 91 shows how the data logically flows through the ECC hardware for a write transaction.

Figure 91. DDR ECC Write Flow



The G-Matrix in Figure 92 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 **AD[63:0]** needs to be XORed together to form the ECC bit. The resulting ECC bits are driven on **CB[7:0]**.



7.3.4.2 DDR ECC Generation for Partial Writes

Figure 92. Intel® 81341 and 81342 I/O Processors G-Matrix (generates the ECC)

	6 3	6 2	6 1	6 0	5 9	5 8	5 7	5 6	5 5	5 4	5 3	5 2	5 1	5 0	4 9	4 8	4 7	4 6	4 5	4 4	4 3	4 2	4 1	4 0	3 9	3 8	3 7	3 6	3 5	3 4	3 3	3 2
CB0																																
CB1																																
CB2																																
CB3																																
CB4																																
CB5																																
CB6																																
CB7																																
	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
CB0																																
CB1																																
CB2																																
CB3																																
CB4																																
CB5																																
CB6																																
CB7																																

When the memory transaction writes less than the data bus width programmed in the SDCR, then the DDR SDRAM Control Block translates the write transaction into a read-modify-write transaction. For a partial write, the DDR SDRAM Control Block calculates the ECC for the modified datum and writes it back. So, when an external unit issues a write cycle with partial data to an DMCU port, the DMCU:

1. Issues a 64-bit (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 64-bit (32-bit) value and ECC.

Note:

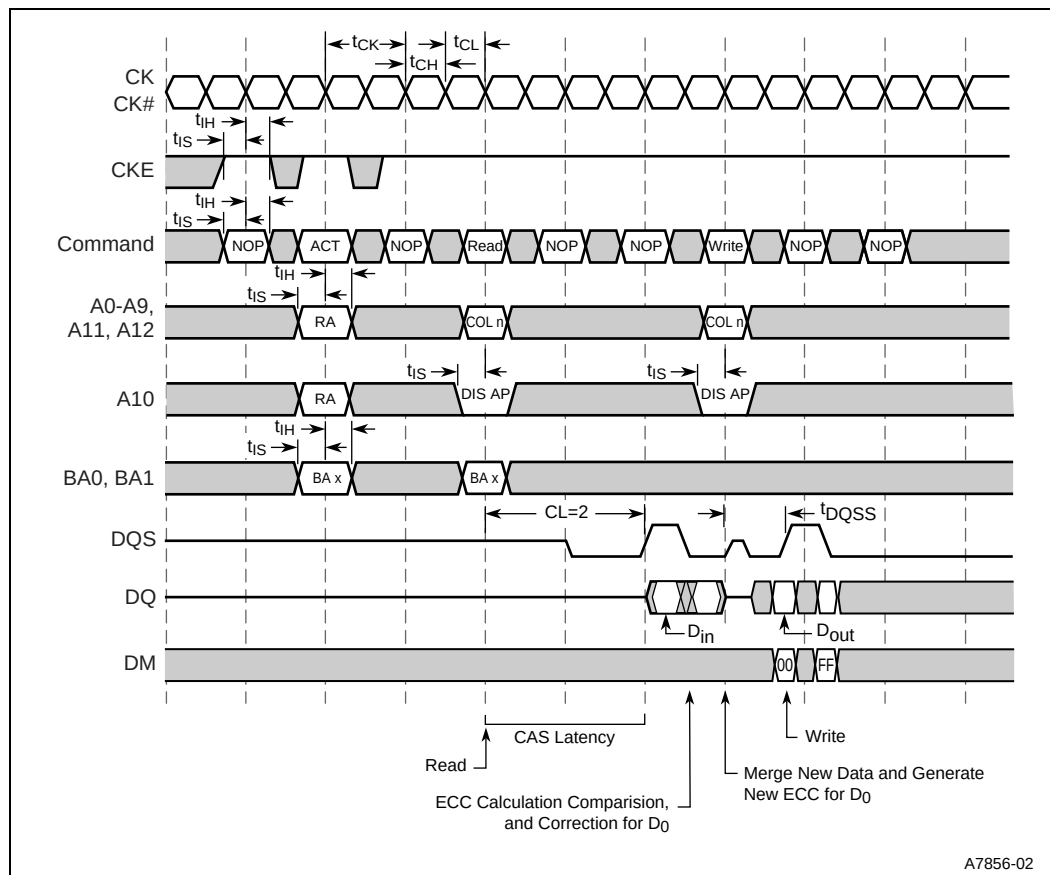
When the DMCU detects a single-bit error during the read, it is corrected BEFORE being merged with the write data so the corrected data is written back to the array. When a multi-bit error is detected, the DMCU causes an interrupt to the core by writing to the MCISR. The memory location is overwritten by the DMCU with the error data but valid ECC, making the contents of memory invalid. For more details on how the DMCU handles error conditions, see [Section 7.5, "ECC Interrupts/Error Conditions" on page 621](#).

[Figure 93](#) shows an example where the data of a write is less than 64-bits wide. The waveform illustrates how the DDR SDRAM Control Block issues a read-modify-write cycle for the data (D_1).

Note:

In 32-bit wide memory and in the 32-bit region in 64-bit wide memory, the DDR SDRAM Control Block still generates 8-bit wide ECC by zero extending the data to 64-bits. A partial write is a write of less than 4-Bytes.

Figure 93. Sub 64-bit DDR SDRAM Write (D_0)





7.3.4.3 DDR ECC Checking

When enabled, the ECC logic uses the following ECC read algorithm. This algorithm corrects the data before it's driven onto the internal bus. The ECC algorithm for a read transaction is:

```
Read 64-bit data and 8-bit ECC
```

```
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 DMCU reads the ECC code from the memory subsystem, it is compared (XORed) with an ECC that the DMCU generates from the data read from the memory. The result is called the syndrome. [Table 370](#) shows how the DMCU decodes the syndrome for DDR SDRAM read cycles.

Table 370. Syndrome Decoding

Error Type	Symptom
None	The syndrome is 0000 0000.
Single-Bit	Use the H-Matrix in Figure 95 to determine which bit the DMCU inverts to fix the error.
Multi-Bit	When the Syndrome does not match an 8-bit value in the H-matrix, the error is uncorrectable

Figure 94 shows how the data flows through the ECC hardware for a read transaction.

Figure 94. DDR ECC Read Data Flow

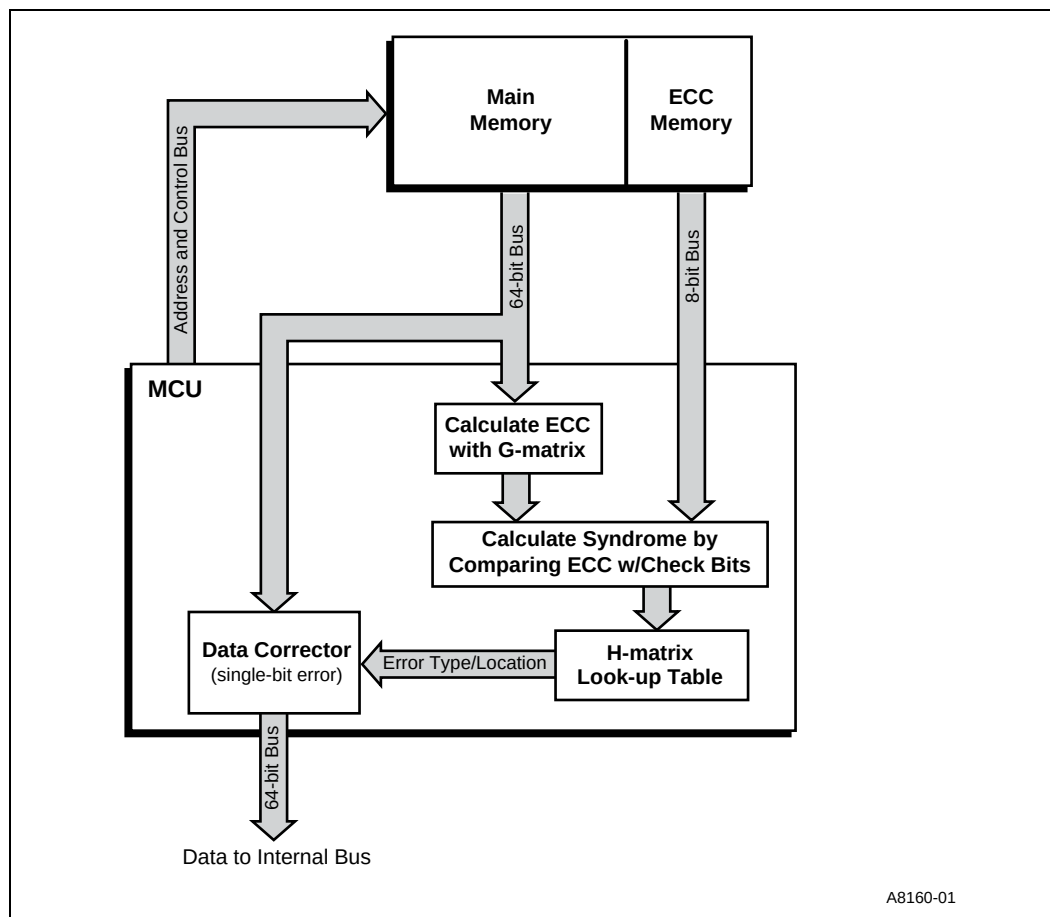




Figure 95 illustrates the H-Matrix used for syndrome decoding. For single-bit errors, the H-Matrix indicates the bit that contains the error and consequently, which bit to fix.

Figure 95. Intel® 81341 and 81342 I/O Processors H-Matrix (indicates the single-bit error location)

	C B 0	C B 1	C B 2	C B 3	C B 4	C B 5	C B 6	C B 7	6 6	6 6	6 6	5 9	5 8	5 7	5 6	5 5	5 5	5 4	5 3	5 2	5 1	5 0	4 9	4 8	4 7	4 6	4 5	4 4	4 4	4 3	4 2	4 1	4 0	3 9	3 8	3 7	3 6
S0	1										1	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	
S2			1									1	1	1	1						1	1	1	1	1	1											
S3				1							1				1											1	1	1	1	1	1	1	1	1	1	1	
S4					1										1						1						1									1	
S5						1					1						1						1								1					1	
S6							1							1													1						1				
S7								1			1	1									1					1							1				
	3 5	3 4	3 3	3 2	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0										0	
S0															1						1											1				1	
S1																		1															1				
S2	1	1	1	1	1			1														1										1					
S3	1	1	1	1	1				1	1											1												1				
S4										1																											
S5																																					
S6																																					
S7	1																																				



Referring to [Figure 94](#), the syndrome bits are created by XORing the data bits as indicated by the appropriate row of the G-Matrix in [Figure 92](#) with the corresponding ECC bit. For example, the DMCU 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 **CB[0]**). The DMCU performs eight such XOR operations (one per syndrome bit).

When decoding the syndrome indicates multi-bit error (see [Table 370](#)), the transaction results in a target-abort for Internal Bus transactions, or a multi-bit error in the BIU for Core transactions. When 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 ECC Error reporting is enabled, the DMCU reports the interrupt in the MCISR and interrupts the core.

When 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 95](#)). For example, when the syndrome was 1100 0001, the error is with bit 0 of **DQ[63:0]**. The DMCU inverts bit 0 before driving the data on **AD[63:0]**.

When error reporting is enabled in the ECCR and the DMCU detects a single-bit or multi-bit error, the DMCU stores the address in ECARx and the syndrome in ELOGx. Then, the DMCU 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 DMCU error conditions and how the MMR registers are affected, refer to [Section 7.5, "ECC Interrupts/Error Conditions" on page 621](#).

Note:

In 32-bit wide memory and in the 32-bit region in 64-bit wide memory, the DDR SDRAM Control Block still generates 8-bit wide ECC by zero extending the data to 64-bits. A partial write is a write of less than 4-Bytes.



7.3.4.4 Scrubbing

Fixing the data error in memory is called scrubbing. The 81341 and 81342 relies on Intel XScale® microarchitecture software to perform the scrubbing. When the DMCU detects an error during a read, the DMCU logs the address where the error occurred and interrupts the core. The core 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, since the DMCU scrubs data during RMW operation. For single-bit errors, the error is fixed by reading the failed location 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 a `ld` instruction and write the data back with a `st` instruction. Software should isolate activity on the memory location to insure animosity.

Note: When scrubbing reads the failed location to fix the single-bit error, a second error is reported. Therefore, software disables single-bit ECC reporting (ECCR[0]) during routine scrubbing. Also, the scrubbing routine maintains partial writes automatically scrub the QWORD aligned location, when it contains a single-bit ECC error.

Multi-bit errors cannot be fixed by the H-Matrix.

7.3.4.4.1 DDR ECC Example Using H-Matrix

When the core writes 1234 5678 9ABC DEF0H to SDRAM memory. The North Internal bus port decoder decodes the address and determines the write being sent to NIBPTQ. NIBPTQ latches the transaction with data 1234 5678 9ABC DEF0H on **AD[63:0]**.

During the next NIBPTQ tenure, this transaction is processed and the DDR SDRAM Control Block receives the data and must calculate the ECC code.

Using G-Matrix in [Figure 92](#), the DDR 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 **CB[7:0]**.

Assuming bit 17 was corrupted in the array, it has then been inverted from 0 to 1.

At some later point in time, the core wishes to read from the same address. The core issues a read transaction which is latched by the NIBPTQ after the North Internal Bus Address Decoder decodes the address and determines the read targets the DDR SDRAM address space. Upon the receipt of 1234 5678 9ABE DEF0H on **DQ[63:0]**, the DDR SDRAM Control Block calculates the syndrome with the G-Matrix in [Figure 92](#). The DDR 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 (**DQ[63:0]**) indicated by the G-Matrix with the check bits (**CB[7:0]**).

In [Table 370](#), when the syndrome is non-zero and matches an H-Matrix value, there is a fixable single-bit error. A syndrome of 52H matches an H-Matrix value ([Figure 95](#)) which indicates bit 17 has an error. The DDR SDRAM Control Block inverts bit 17 prior to returning corrected data on **AD[63:0]**. DMCU returns 1234 5678 9ABC DEF0H on **AD[63:0]**.

Assuming this was the first error, the DMCU records the address where the error occurred in ECAR0 and error type in ELOG0. When error reporting is enabled in the ECCR, the DMCU writes a 1 to DMCISR[0] which generates an interrupt to the core. A software interrupt handler scrubs the array and fixes error in bit 17. Unless more errors occur, future reads from this location do not result in an error.



7.3.4.5 DDR ECC Disabled

When software disables ECC, the DMCU does generate the ECC byte for writes, but does not check the ECC byte for reads.

7.3.4.6 DDR ECC Testing

Section 7.3.4.4, “Scrubbing” on page 615 explains how the software is responsible for correcting an error in the memory array once it has been detected by the ECC logic. The DMCU 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 DMCU reads the address later, the ECC mismatches and the error condition occurs (see Section 7.5, “ECC Interrupts/Error Conditions” on page 621).

7.3.5 Memory Regions

The DMCU supports two independent memory regions:

- Memory Mapped Register (MMR) Space
- DDR SDRAM Memory Space

The MMR memory space ranges from offset +1800H through +19FFH of the MMR Base address registers. Software programs the DDR SDRAM memory space by providing a base address in SDBR and SDUBR, the size and number of DDR SDRAM banks in SBSR, and the size of the 32-bit region in S32SR when desired.

7.3.6 DDR SDRAM Clocking

The DMCU provides 6 clocks, three positive (**M_CK[2:0]**) and three negative (**M_CK[2:0]#**), to the DDR SDRAM memory subsystem at the selected DDR SDRAM command rate. The 72-bit 2-bank unbuffered DDR SDRAM DIMM specification requires 6 clocks to distribute the loading across eighteen x8 DDR SDRAM components.



7.4 Power Failure Mode

The 81341 and 81342 is an I/O processor used in server applications including networking and storage. Specifically, the storage applications supported utilize the 81341 and 81342 as the IOP for a SCSI RAID disk subsystem and the local memory is used for disk caching. The local memory is used for the temporary storage of disk writes which greatly improves disk performance.

While the host assumes all written data is stored on the non-volatile disk subsystem, the IOP must ensure that eventually all the data in the disk cache is actually stored onto disk.

The power supply could fail to provide power to the I/O subsystem in the case of a power outage or a failed power supply. It is imperative that the cached data within the IOPs local memory is not lost. When power fails, the local memory subsystem must remain powered with a battery backup and some agent must continue to refresh at the appropriate interval specified by the memory component datasheet.

This section defines a mechanism with which the 81341 and 81342 memory controller ensures that the data within local memory is not lost during a power failure.

7.4.1 Theory of Operation

DDR SDRAM technology provides a simple way of enabling data preservation through the **self-refresh** command. This command is issued by the memory controller and the DDR SDRAM refreshes itself autonomously with internal logic and timers. The **self-refresh** command is defined in [Table 364](#).

The DDR SDRAM device remains in self-refresh mode as long as:

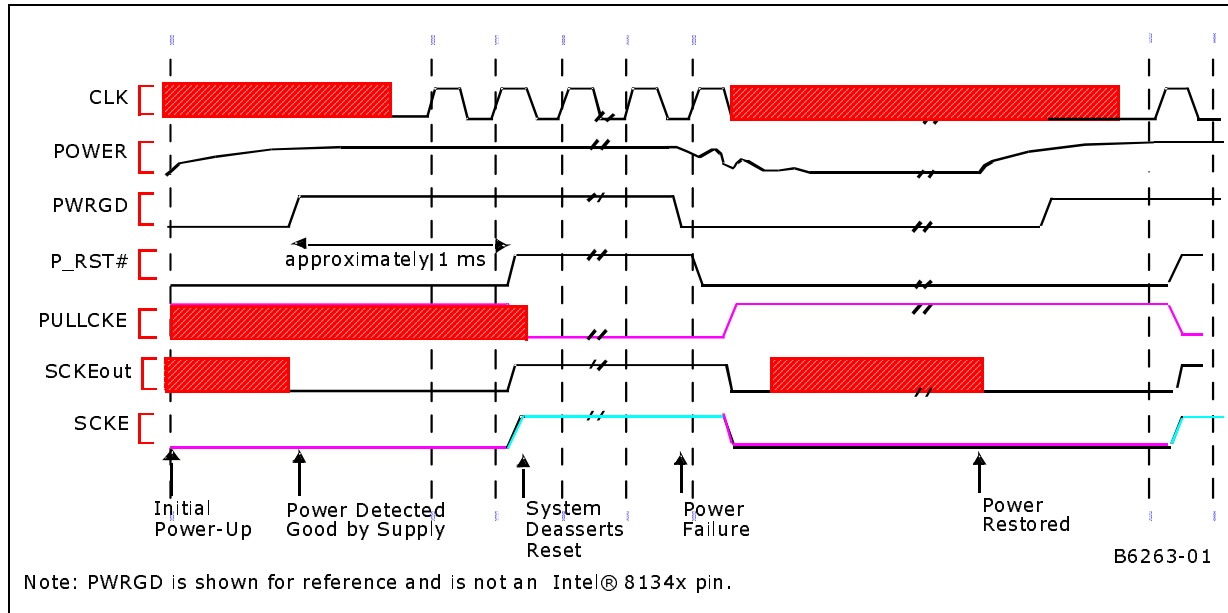
- The device continues to be powered.
- **CKE** is held low until the memory controller is ready to control the DDR SDRAM once again.

Power to the DDR SDRAM subsystem is ensured with an adequate battery backup and a reliable method for switching between system power and battery power. The memory controller is responsible for deasserting **CKE[1:0]** when issuing the **self-refresh** command but while power gradually drops, **CKE[1:0]** must remain deasserted regardless of the state of V_{CC} powering the 81341 and 81342.

7.4.2 Power Failure Sequence

Figure 96 illustrates the sequence of events during a power failure as defined by *PCI Local Bus Specification*, Revision 2.2.

Figure 96. Power Failure Sequence



7.4.2.1 Power Failure Impact on the System

Upon initial power-up a power supply provides the appropriate voltage to the system. The voltage level increases at a rate that is dependent on the type of power supply used and the components in the system. These variables are not certain, so the power supply often provides a signal called **PWRGD** which indicates the time when the voltage has reached a reliable level. The power supply deasserts **PWRGD** when the voltage level drops below a certain minimum threshold.

PCI Local Bus Specification, Revision 2.2 indicates that once **PWRGD** is deasserted, the PCI reset pin (**P_RST#**) is asserted in order to float the output buffers. In the specification T_{fail} is defined as the time when **P_RST#** is asserted in response to the power rail going out of specification. T_{fail} is the minimum of:

- 500 ns from either power rail going out of specification (exceeding specified tolerances by more than 500mV).
- 100 ns from the 5V rail falling below the 3.3V rail by more than 300mV.

7.4.2.2 System Assumptions

This proposal makes specific assumptions about the system behavior during a power failure. When the below assumptions are not insured, it is the vendor responsibility to ensure them.

1. **P_RST#** is asserted to the 81341 and 81342 when there is at least 2 μ s of reliable power remaining. This is required so that the memory controller can execute its power-failure state machine in response to the assertion of **P_RST#**.



7.4.3 Memory Controller Response to P_RST#

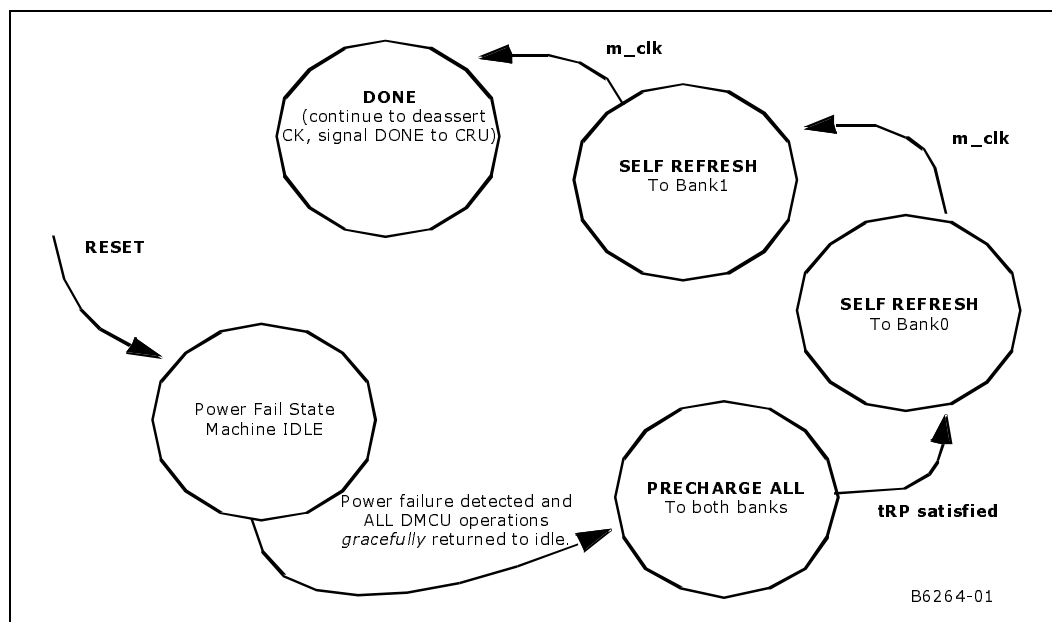
The memory controller assumes a power failure condition whenever **P_RST#** is asserted. **P_RST#** assertion following an initial power up sequence results in the following sequence of events:

1. Clock and Reset Unit (CRU) requests that DMCU run the power fail sequence depicted in Figure 96. The DMCU takes the following steps to execute the power fail sequence:
 - a. Gracefully terminate the current transaction.
 - b. Deactivate all DDR SDRAM leaves with the **precharge-all** command.
 - c. After Trp, the DMCU issues a **self-refresh** command to the DDR SDRAM devices one bank at a time and continue to deassert **CKE[1:0]**.
2. The DMCU notifies the CRU that the power fail sequence has completed on the memory bus.
3. The CRU then asserts the internal bus reset to reinitialize all internal bus agents including the DMCU.

Note: Internal bus reset is asserted in response to the assertion of **P_RST#**. When **P_RST#** indicates a true power failure, then battery-backup power is supplied to the DDR SDRAM array. A true power failure must be determined by external circuitry based on **P_RST#** and also by other external logic monitoring the system power.

Refer to Figure 97 for a high-level state machine representation illustrating the memory controller's behavior during a power failure condition.

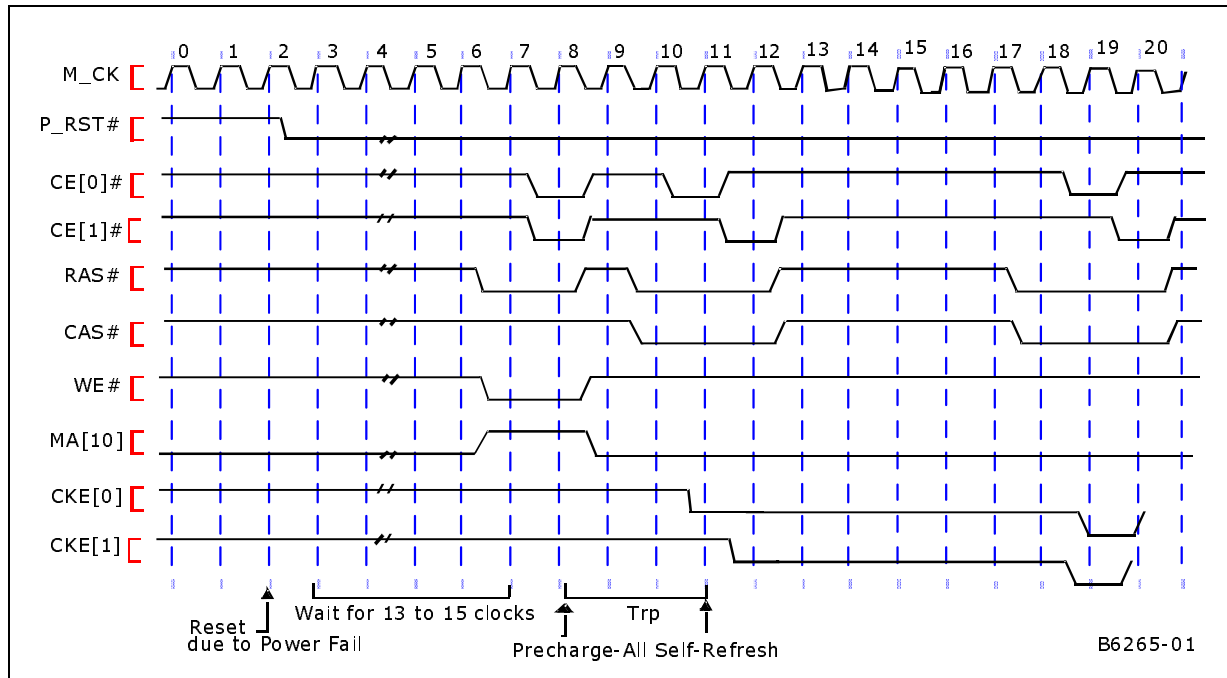
Figure 97. Power Failure State Machine



Note: Following the request from the CRU to run the power fail sequence, any data that is in the DMCUs 1024 byte posted write buffer is discarded.

Figure 98 illustrates the DDR SDRAM waveforms after the assertion of **P_RST#** during a true power failure.

Figure 98. Power Failure Sequence



CKE[1:0] must be held low throughout the power-down period. The memory controller drives it low initially with the **self-refresh** command, but an external pull-down is required to continually drive it low when the 81341 and 81342 loses power. External logic ensures that **CKE[1:0]** is held low after the memory controller initially deasserts it. Likewise, the external logic must stop driving **CKE[1:0]** low once **P_RST#** is deasserted by the system.

Due to the high loading on **CKE** and the requirement of PC200 DDR operation, the memory controller must drive two copies to the DDR SDRAM DIMM. The board layout distributes the two **CKE[1:0]** signals between the two DDR SDRAM banks equally.

As long as the DDR SDRAM memory subsystem is powered with a battery source and **CKE[1:0]** is held low, the DDR SDRAM preserves its memory image.

When power is restored, the system asserts **P_RST#** to the 81341 and 81342. While the 81341 and 81342 is reset, **CKE[1:0]** is held low by memory controller. After **P_RST#** is deasserted (and subsequently, internal bus reset is deasserted), the 81341 and 81342 must be re-initialized to reset the DDR SDRAM memory subsystem operating parameters. The first step of DDR SDRAM initialization sequence re-asserts **CKE[1:0]** to ones and the memory controller resumes refreshing. DDR SDRAM initialization sequence does not affect memory contents. For more details about the DDR SDRAM initialization sequence, refer to [Section 7.3.3.7, "DDR SDRAM Initialization"](#) on page 594.

Note:

The power failure mechanism in the memory controller is not responsible for maintaining the 81341 and 81342 state. The purpose of this mechanism is to maintain the memory so that any data cached in the local memory can be flushed once power is restored. Any data queued within the 81341 and 81342 components (DMCU, ATU, DMAs, etc.) is lost.



7.5 ECC Interrupts/Error Conditions

The DMCU has two conditions which require intervention from the Intel XScale® microarchitecture. When a single-bit error is detected during a read cycle, the DMCU corrects the data returned but software needs to fix the error in the memory array. When a multi-bit error is detected, the core decides how to handle the condition. For all ECC errors, the DMCU records the requester of the transaction resulting in the error in DELOGx[23:16] and interrupts the core.

When the DMCU detects an ECC error during a read or write cycle²³, DMCISR[0] or DMCISR[1] is set to 1. Whenever the DMCU toggles one of the DMCISR bits from 0 to 1, an interrupt is generated to the core.

Table 371 shows how the DMCU responds to error conditions.

Table 371. DMCU Error Response

Error Type	DMCU Action ¹
Single-Bit during a read or write	Fix Error (when ECC error correction enabled in the DECCR)
Multi-bit during a read	Target Abort the Internal Bus or direct port transaction.
Multi-bit during a write	New ECC is generated with bad data and written to DDR SDRAM array. Data location is no longer valid.

Notes:

1. The ECC Enable bit in the DECCR must be set in order for these actions to occur.

Note: When ECC reporting is enabled with DECCR[1] or DECCR[0] and an ECC error occurs, DMCISR[1] or DMCISR[0] is set and DELOGx/DEARx logs the error in addition to Table 371 actions.

²³Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See Section 7.3.4.1, "DDR ECC Generation" on page 608 for details.

7.5.1 Single-Bit Error Detection

When enabled, the DMCU interrupts the core when the ECC logic detects a single-bit error by setting the appropriate bit in the DMCISR register. The core knows the interrupt was caused by a single-bit error by polling the DELOG0 or DELOG1 register. The DDR SDRAM Control Block ensures that correct data is returned but the interrupt handler is responsible for scrubbing the error in the array (refer to [Section 7.3.4.4, “Scrubbing” on page 615](#)).

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 DMCU.
- The DMCU fixes the error prior to returning the data.
- The DMCU clears DELOG0[8] indicating a single-bit error.
- The DMCU records the requester of the transaction that resulted in an error in DELOG0[23:16]
- The DMCU loads DELOG0[7:0] with the syndrome that indicated the error.
- The DMCU loads DELOG0[31:28] and DEAR0[31:2] with address where the error occurred. In addition, when an error occurs during an ADMA transfer, the context address is loaded in DECUAR0 and DECAR0.
- Core needs to scrub error in array, DMCU sets DMCISR[0] to 1 (assuming not already set).
 - Setting any bit in the DMCISR causes an interrupt to the core.
- Software polls the interrupt status register. Bit 0 set to 1 indicates that the first error has occurred.
- Software polls DELOG0 and DEAR0 and scrubs the error at the location specified by DEAR0.
- Software writes a 1 to DMCISR[0] thereby clearing it.

When software does not perform error scrubbing, the probability of an unrecoverable multi-bit error increases for the memory location containing the single-bit error.

DEARx and DELOGx remain registered until software explicitly clears them.

When a second error occurs before software clears the first by resetting DMCISR[0] or DMCISR[1], the error is recorded in the remaining DELOGx/DEARx register. When none are available, the error is not logged but the DMCU carries out the action described in [Table 371](#).



7.5.2 Multi-bit Error Detection

When a multi-bit error occurs during a read or write transaction and error reporting is enabled, the DMCU sets DMCISR[0] or DMCISR[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 DELOGx registers.

When DMCU detects a multi-bit error during a read cycle and ECC calculation is enabled in the DECCR, the DMCU target aborts the transaction in the NIBPTQ or SIBPTQ, indicating to the internal bus masters that an unrecoverable error has been detected. For direct port transactions issued by the Application DMAs, when a multi-bit error is detected during a read cycle, the DMCU signals a target abort to the agent. The DMCU records the error type in DELOGx and the address in DEARx.

When DMCU detects a multi-bit error during a write²⁴ cycle and error reporting is enabled in the DECCR, the DMCU records the first multi-bit error by programming DELOGx and DEARx. The DMCU generates new ECC with the data before sending it on **DQ[63:0]** so the contents of memory after the read-modify-write cycle are corrupted with correct ECC.

When a second error occurs before software clears the first by resetting DMCISR[0] or DMCISR[1], the error is recorded in the remaining DELOGx/DEARx register. When none are available, the error is not logged but the DMCU carries out the action described in [Table 371](#).

It is interrupt handler responsibility to decide how to handle this error condition and clear the DMCISR.

²⁴Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See [Section 7.3.4.1, “DDR ECC Generation” on page 608](#) for details.



7.6 Parity Interrupts/Error Conditions

When a data parity error is detected on any of the DMCU ports, the DMCU records the requesting port that caused the parity error in the DPCSR[19:16] and interrupts (when enabled) the core. Refer to [Section 7.8.15, "DDR Parity Control and Status Register — DPCSR" on page 644](#).

When the DMCU detects a parity error, the DMCISR[8] is set to 1. Whenever the DMCU toggles the DMCISR[8] bit from 0 to 1, an interrupt is generated to the core.

Note: The DMCU generates a single hardware interrupt (DDR Memory Controller Unit Error Interrupt Pending) to the Interrupt Controller Unit (ICU) for either a ECC error or a parity error. ECC and parity error interrupts must be enabled for the interrupt to be generated.

7.7 Reset Conditions

Software must issue an initialization sequence as defined in [Section 7.3.3.7, "DDR SDRAM Initialization" on page 594](#). After the initialization sequence, the DDR 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 when ECC is enabled.

While **P_RST#** is asserted, the DMCU initializes its MMR registers to the states defined in [Section 7.8, "Register Definitions" on page 625](#).

Note: The operation of any memory transactions are not insured when **P_RST#** is asserted.



7.8 Register Definitions

A series of configuration registers control the DMCU. Software determines the DMCU status by reading status registers. [Table 372](#) lists all DMCU registers detailed further in proceeding sections.

Note: Constant polling of DMCU MMRs should be avoided since it can result in inducing long latencies in peripheral unit DDR SDRAM transactions and therefore may negatively impact performance.

Table 372. Memory Controller Registers

Section, Register Name — Acronym (Page)
Section 7.8.1, "SDRAM Initialization Register — SDIR" on page 627
Section 7.8.2, "SDRAM Control Register 0 — SDCR0" on page 628
Section 7.8.3, "SDRAM Control Register 1 — SDCR1" on page 630
Section 7.8.4, "SDRAM Base Register — SDBR" on page 632
Section 7.8.5, "SDRAM Upper Base Register — SDUBR" on page 633
Section 7.8.7, "SDRAM Bank Size Register — SBSR" on page 635
Section 7.8.8, "SDRAM 32-bit Region Size Register — S32SR" on page 637
Section 7.8.9, "DDR ECC Control Register — DECCR" on page 638
Section 7.8.10, "DDR ECC Log Registers — DELOG0, DELOG1" on page 639
Section 7.8.11, "DDR ECC Address Registers — DEAR0, DEAR1" on page 641
Section 7.8.12, "DDR ECC Context Address Registers — DECAR0, DECAR1" on page 642
Section 7.8.13, "DDR ECC Context Upper Address Registers — DECUAR0, DECUAR1" on page 643
Section 7.8.14, "DDR ECC Test Register — DECTST" on page 643
Section 7.8.15, "DDR Parity Control and Status Register — DPCSR" on page 644
Section 389, "DDR Parity Address Register — DPAR" on page 645
Section 390, "DDR Parity Upper Address Register — DPUAR" on page 645
Section 391, "DDR Parity Context Address Register — DPCAR" on page 646
Reserved.
Section 392, "DDR Parity Context Upper Address Register — DPCUAR" on page 646
Reserved.
Section 7.8.20, "DDR Memory Controller Interrupt Status Register — DMCISR" on page 647
Section 7.8.21, "DMCU Port Transaction Count Register — DMPTCR" on page 648
Section 7.8.23, "Frequency Register — RFR" on page 651



Table 373. Memory Controller Pad Registers

Section, Register Name — Acronym (Page)
Section 7.8.24, “DDR RCOMP Control Register — DRCR” on page 652
Section 7.8.25, “RCOMP Pad Drive Strength Select Register — RPDSR” on page 654
Section 7.8.26, “DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR” on page 655
Section 7.8.27, “DQ Pad Drive Strength Manual Override Values Register — DQPDSR” on page 656
Section 7.8.28, “MA Pad Drive Strength Manual Override Values Register — MAPDSR” on page 657
Section 7.8.29, “MCLK Pad Drive Strength Manual Override Values Register — MPDSR” on page 658
Section 7.8.30, “CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR” on page 659
Section 7.8.31, “DLL Delay Register 0 — DLLR0” on page 660
Section 7.8.32, “DLL Delay Register 1 — DLLR1” on page 661
Section 7.8.33, “DLL Delay Register 2 — DLLR2” on page 662
Section 7.8.34, “DLL Delay Register 3 — DLLR3” on page 663
Section 7.8.35, “DLL Delay Register 4 — DLLR4” on page 664
Section 7.8.36, “DLL Delay for Receive Enable Register — DLLRCVER” on page 665



7.8.1 SDRAM Initialization Register — SDIR

The DDR SDRAM Initialization Register (SDIR) is responsible for programming the operation of the DDR SDRAM device state machines. The SDIR provides a method for software to execute the DDR SDRAM initialization sequence (see [Section 7.3.3.7, “DDR SDRAM Initialization” on page 594](#)). [Table 365](#) provides the encoding that software can use to initialize the SDRAM.

Note: Software is responsible to provide all the parameters required for SDRAM DDR commands, Mode Register and Extended Mode Register using the SDIR bit fields. For example, parameters are not supplied by other memory-mapped registers.

Table 374. DDR SDRAM Initialization Register — SDIR



SDRAM Control Registers (SDCR[1:0]) are responsible for programming operation of the DDR SDRAM state machines. The SDCR0 specifies the DIMM type, data bus width, and some SDRAM timing parameters required by the DDR SDRAM state machine as defined in ["DDR SDRAM Initialization" on page 594](#) and ["DDR SDRAM Mode Programming" on page 601](#). The remaining SDRAM timing parameters required by the DDR SDRAM state machine are set in SDCR1.

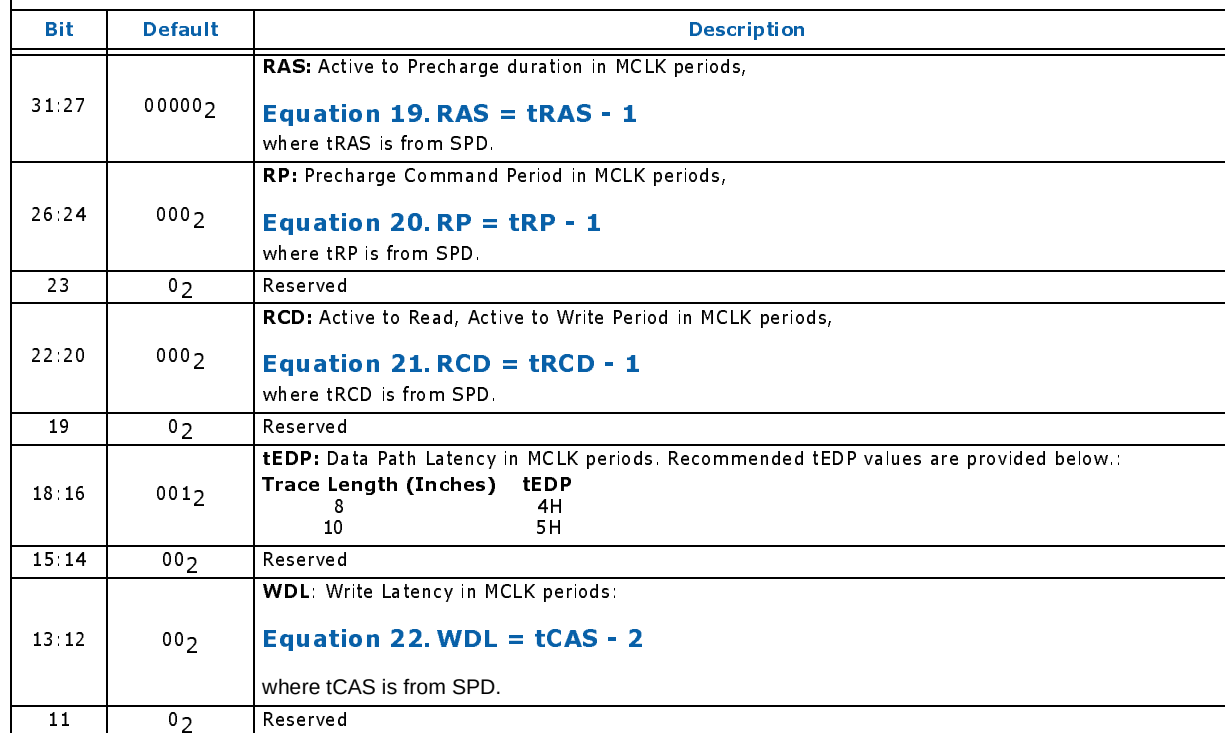
Table 375. DDR SDRAM Control Register 0 — SDCR0 (Sheet 1 of 2)

Table 375. DDR SDRAM Control Register 0 – SDCR0 (Sheet 2 of 2)

Intel XScale® microarchitecture Local Bus Address offset +1804H

Attribute Legend:
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

Bit	Default	Description
10:08	000 ₂	CAS: Indicates the CAS Latency used by the Memory Controller state machine. Equation 23. CAS = tCAS - 1 where tCAS is from SPD. 000 = Reserved. 001 = Reserved. 010 = 3 MCLK periods (DDR2 Type only) 011 = 4 MCLK periods (DDR2 Type only) 100 = 5 MCLK periods (DDR2 Type only) All other values are reserved and not supported. Note: When ODT is enabled CAS Latency must be equal to or greater than 4.
07	0 ₂	Reset read FIFO. Writing a 0 to this bit causes the FIFO to be reset. 0 = Reset FIFO 1 = Normal Operation
06	0 ₂	Internal Bus Address Map Control — This bit controls how the internal bus address is mapped to the Bank Address/Row Address/Column Address of the DDR2 SDRAM Memory. Refer to Section 7.3.3.2, "DDR SDRAM Addressing" on page 575 for more details.
05:04	00 ₂	ODT Termination Value: Determines the termination value of the On Die Termination for both Banks (controlled by ODT[1:0]). Applies to DDR2 SDRAM memory type only. 00 Disabled 01 75 ohm 10 150 ohm 11 reserved
03	0 ₂	Reserved.
02	0 ₂	DDR Type. Only reflects DDR2 for 81341 and 81342.
01	0 ₂	Data Bus Width: Indicates the width of the data bus. See Section 7.3.3.4, "32-bit Data Bus Width" on page 588. 0 = 64 bits 1 = 32 bits
00	0 ₂	DIMM Type: Selects unbuffered or registered DIMM operating modes for the DMCU. 0 = Unbuffered 1 = Registered (must always be set for 81341 and 81342) Note: The DMCU on 81341 and 81342 must always be programmed to operate in Registered mode. Hence, this bit must be set. When interfacing with unbuffered DIMM type, software must set bit DLLRCVER[16]. Refer to "DLL Delay for Receive Enable Register — DLLRCVER" on page 665.



The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDR SDRAM state machines as defined in “[DDR SDRAM Initialization](#)” on page 594 and “[DDR SDRAM Mode Programming](#)” on page 601. The SDCR1 specifies the remaining SDRAM timing parameters required by the DDR SDRAM state machine not specified in SDCR0.

Warning: SPD values read must be converted into MCLK Periods.

Table 376. DDR SDRAM Control Register 1 - SDCR1 (Sheet 1 of 2)

Intel® 81341 and 81342 I/O Processors
Developer's Manual
630

**Table 376. DDR SDRAM Control Register 1 - SDCR1 (Sheet 2 of 2)**

Bit	Default	Description
18	0 ₂	Reserved
17:12	000000 ₂	RFC: Refresh to Active and Refresh to Refresh period in MCLK periods Equation 27. $RFC = tRFC - 1$ where tRFC is from SPD.
11:09	000 ₂	WR: Write Recovery time in MCLK periods. Equation 28. $WR = tWR$ where tWR is from SPD. Note: This parameter is not used by the DMCU.
08:04	00000 ₂	RC: Active to Active and Active to Refresh period in MCLK periods. Equation 29. $RC = tRC - 1$ where tRC is from SPD.
03:00	0000 ₂	WTRD: Write to Read turnaround period in MCLK periods. Equation 30. $WTRD = tCAS - 1 + (BL/2) + tWTR$ where BL = 4, tCAS and tWTR are from SPD.



7.8.4 SDRAM Base Register — SDBR

This register indicates six bits of the beginning address (base address) of SDRAM space. The SDRAM is addressed using a 36-bit address. This register is used in conjunction with the [Section 7.8.5, SDRAM Upper Base Register — SDUBR](#). See [Section 7.3.3.3, “DDR SDRAM Bank Sizes and Configurations”](#) on page 578 for usage details. There can be two contiguous physical banks defined by SBSR in the DDR SDRAM subsystem starting at this address.

Note: DDR2 SDRAM memory space must **never** cross a 4 Gbyte boundary.

Note: This register should be read back after being written, before the Intel XScale® microarchitecture performs transactions which address the DDR SDRAM.

Warning: The SDBR is internally used by the DDR MCU to create internal control signal. After initializing the DDR MCU for normal operation, when the user decides to reconfigure the DDR MCU at a later time by clearing SDBR, the user must wait for a period of inactivity on both the north and south internal buses before accessing the DDR MCU again to allow ample time for the new values to settle.

Table 377. SDRAM Base Register — SDBR

Bit	Default	Description																													
31:27	0	SDRAM Base Address: These bits define part of the upper bits of the DDR SDRAM base address.																													
26:00	0	Reserved																													



7.8.5 SDRAM Upper Base Register — SDUBR

This register indicates the upper four bits of the beginning address (base address) of SDRAM space. The SDRAM is addressed using a 36-bit address. This register is used in conjunction with the [Section 7.8.4, SDRAM Base Register — SDBR](#). Refer to See [Section 7.3.3.3, “DDR SDRAM Bank Sizes and Configurations” on page 578](#) for usage details. There can be two contiguous physical banks defined by SBSR in the DDR SDRAM subsystem starting at this address.

Note: DDR SDRAM memory space must **never** cross a 4 Gbyte boundary.

Note: This register should be read back after being written, before the Intel XScale® microarchitecture performs transactions which address the DDR SDRAM.

Warning: The SDUBR is internally used by the DDR MCU to create internal control signal. After initializing the DDR MCU for normal operation, when the user decides to reconfigure the DDR MCU at a later time by clearing SDUBR, the user must wait for a period of inactivity on both the north and south internal buses before accessing the DDR MCU again to allow ample time for the new values to settle.

Table 378. SDRAM Upper Base Register — SDUBR

Bit	Default	Description
31:04	0	Reserved
03:00	0	SDRAM Upper Base Address: Define part of the upper bits of the DDR SDRAM base address.



Secondary SDRAM Base Register - SSDBR

This register indicates six bits of the beginning address (base address) of a Secondary SDRAM Memory Window of SDRAM space. The secondary memory window is addressed using a 32-bit address (upper 4 bits of the Intel XScale® microarchitecture 36-bit address are zero). The secondary memory window defined by SSDBR[31:26] and SBSR[21:16] addresses a range of the SDRAM space. If the SDRAM space is defined to be less than 4 GBytes, the user must ensure that the secondary memory window is defined so that its address range is mapped within the SDRAM space. Refer to [Figure 82, “Secondary DDR SDRAM Window Memory Map” on page 582](#) which shows the valid address range for the secondary memory window relative to the SDRAM space.

Note: The Secondary SDRAM memory space must **never** be larger than the total SDRAM size defined by the SBSR, or 2 GByte, whichever is smaller.

Note: This register reads back after being written, before the Intel XScale® microarchitecture performs transactions which address the DDR SDRAM.

Table 379. Secondary SDRAM Base Register - SSDBR

Intel XScale® microarchitecture Local Bus
Address offset
+1980H

Attribute Legend:
RW = Read/Write
RV = Reserved
PR = Preserved
RS = Read/Set
RC = Read Clear
RO = Read Only
NA = Not Accessible



7.8.7 SDRAM Bank Size Register — SBSR

This register indicates size and number of DDR SDRAM banks. These parameters are used to calculate total DDR SDRAM memory size supported. The total DDR SDRAM size calculated is then used to program the Base Address Registers. For example, the Base address Registers must align to the total size of DDR memory supported. This register is also used to indicate the bank boundaries for address decoding and bank select signals generation. Bit[0] of this register also provides the DDR technology type.

Note: DDR SDRAM memory space must **never** cross a 4 Gbyte boundary.

Note: This register is read back after being written, before the Intel XScale® microarchitecture performs transactions which address the DDR SDRAM.

Warning: SPD values read must be converted into MCLK Periods.

Warning: SDRAM Bank Size and Number of DDR Banks fields are internally used by DDR MCU to create internal control signals. After initializing DDR MCU for normal operation, when the user decides to reconfigure DDR MCU at a later time by clearing any of these fields, the user must wait for a period of inactivity on both the north and south internal buses before accessing the DDR MCU again to allow ample time for the new values to settle.

Table 380. SDRAM Bank Size Register — SBSR (Sheet 1 of 2)

		IOP Attributes																PCI Attributes																															
		na na na na na na na na na na na na na na na na na na na na na na na na na na																																															
		Intel XScale® microarchitecture Local Bus Address offset +1814H																Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																																															
31:27	00000 ₂	SDRAM Bank Size: Defines the size of a single DDR SDRAM bank. 00000 ₂ = Empty 00001 ₂ = 128 MB 00010 ₂ = 256 MB 00100 ₂ = 512 MB 01000 ₂ = 1 GB all other values are reserved. Note: 128 MBytes memory size is only possible on 81341 and 81342 when supporting a 32-bit data bus. For example, minimum memory size supported with a 64-bit data bus is 256 MBytes.																																															
26:12	000_0H	Reserved																																															
21:16	00_0000 ₂	Secondary SDRAM Bank Size: Defines the size of the Secondary DDR SDRAM window addressed by the Secondary SDRAM Base Register. 00 0000 ₂ = Secondary window disabled 00 0001 ₂ = 64 MB 00 0010 ₂ = 128 MB 00 0100 ₂ = 256 MB 00 1000 ₂ = 512 MB 01 0000 ₂ = 1 GB 10 0000 ₂ = 2 GB all other values are reserved.																																															
15:12	0H	Reserved																																															
11:08	0000 ₂	RMW: Read to Write turnaround period in MCLK periods for Read-Modify-Write. Equation 31. $RMW = t_{CAS} + (BL/2) + t_{EDP}$ where BL = 4, tEDP = (Refer to SDCR0[18:16]), and tCAS is from SPD.																																															



Table 380. SDRAM Bank Size Register — SBSR (Sheet 2 of 2)

312824201612840 IOP Attributes rw/rw/rw/rw/rw/rv/rv/rv/rv/rv/rw/rw/rw/rw/rw/rv/rv/rv/rv/rw/rw/rw/rw/rw/rv/rv/rv/rv/rw/rw/rw PCI Attributes na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na/na Intel XScale® microarchitecture Local Bus Address offset +1814H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																															
Bit	Default	Description																													
07:06	00 ₂	Reserved																													
05:03	000 ₂	Reserved.																													
02	0 ₂	Number of DDR Banks — This field indicates the number of DDR Banks supported. Up to two DDR banks are supported. 0 ₂ = 2 Banks 1 ₂ = 1 Bank																													
01:00	00 ₂	SDRAM Technology: Defines the memory subsystem technology. 00 ₂ = 512 Mbit 01 ₂ = 1 Gbit 10 ₂ = 2 Gbit 11 ₂ = Reserved																													



7.8.8 SDRAM 32-bit Region Size Register — S32SR

Defines the size of the 32-bit region located at the base of SDRAM Bank 0. This register must be programmed with a size that is less than or equal to one half of the size of DDR SDRAM Bank 0. Sizes are limited to binary sizes with a minimum size of 1MB. Also, the DDR SDRAM type must be 64-bit, as defined by “SDRAM Control Register 0 — SDCR0” on page 628. See also “SDRAM Bank Size Register — SBSR” on page 635 and Section 7.3.3.3, “DDR SDRAM Bank Sizes and Configurations” on page 578.

This register is read back after being written, before the Intel XScale® microarchitecture performs transactions which address the DDR SDRAM.

Table 381. DDR SDRAM 32-bit Region Size Register — S32SR

Intel XScale® microarchitecture Local Bus Address offset +1818H Attribute Legend: RW = Read/Write, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:30	00 ₂	Reserved
29:20	000H	32-bit Region Size — indicates the size of the 32-bit region at the base of Bank0 when a 64-bit data bus width is implemented. This size is also the size of the invalid region adjacent to the 32-bit region. 00 0000 0000 = no 32-bit region defined 00 0000 0001 = 1MB 00 0000 0010 = 2MB 00 0000 0100 = 4MB 00 0000 1000 = 8MB 00 0001 0000 = 16MB 00 0010 0000 = 32MB 00 0100 0000 = 64MB 00 1000 0000 = 128MB 01 0000 0000 = 256MB 10 0000 0000 = 512MB all other values are reserved.



7.8.9 DDR ECC Control Register — DECCR

This register programs the DMCU error correction and detection capabilities. The configuration depends on the application needs, but a typical configuration is:

- DDR ECC Mode Enabled
- Enable multi-bit error reporting
- Disable single-bit error reporting
- Enable single-bit error correcting

For more details, see [Section 7.3.4, “DDR Error Correction and Detection”](#) on page 607 and [Section 7.5, “ECC Interrupts/Error Conditions”](#) on page 621.

Table 382. DDR ECC Control Register — DECCR

Bit	Default	Description	
31:04	000 0000H	Reserved	
03	0 ₂	ECC Enabled: Enables ECC calculation and generation. 0 = ECC Disabled 1 = ECC Enabled	
02	0 ₂	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	
01	0 ₂	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	
00	0 ₂	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	



7.8.10 DDR ECC Log Registers — DELOG0, DELOG1

The DDR ECC Log Registers are responsible for logging the error types detected on the local memory bus. Two errors are 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 does not match an entry in the H-Matrix and thus, is uncorrectable (see [Table 370, “Syndrome Decoding” on page 611](#)).

The error recorded in DELOG0 corresponds to the address in DEAR0, and the error logged in DELOG1 corresponds to the address in DEAR1. The upper four bits of the 36-bit ECC address is logged into this register.

The ELOGx registers comprise read-only bits and only have meaning when MCISR[0] or MCISR[1] is non-zero. For more details on error handling, see [Section 7.3.4, “DDR Error Correction and Detection”](#) on page 607.

Table 383. DDR ECC Log Registers — DELOG0, DELOG1 (Sheet 1 of 2)

IOP Attributes

PCI Attributes

Error #

Intel XScale® microarchitecture Local Bus Address offset

Attribute Legend:

RW = Read/Write
 RC = Read Clear
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible

Bit	Default	Description																						
31:28	0H	Upper ECC Address — The upper 4 bits of the 36-bit ECC Address is stored in this 4-bit field when an ECC error is logged. The lower 29 bits of the address are logged in the DDR ECC Address Registers (DEARx).																						
27:24	0000 ₂	Reserved																						
23:20	0H	ECC Error Requester: Indicates the requester of the logged error: <table> <tr> <th>Internal Bus Requester ID</th> <th>Requester Name</th> </tr> <tr> <td>0000₂</td> <td>Reserved</td> </tr> <tr> <td>0001₂</td> <td>Intel XScale® microarchitecture 0 (coreID0)</td> </tr> <tr> <td>0010₂</td> <td>Intel XScale® microarchitecture 1 (coreID1)</td> </tr> <tr> <td>0011₂</td> <td>ATUX</td> </tr> <tr> <td>0100₂</td> <td>ATUE</td> </tr> <tr> <td>0101₂</td> <td>Application DMAs</td> </tr> <tr> <td>0110₂</td> <td>Reserved</td> </tr> <tr> <td>0111₂</td> <td>Reserved</td> </tr> <tr> <td>1000₂</td> <td>Reserved</td> </tr> <tr> <td>1001₂</td> <td>SMBus</td> </tr> </table> All other IDs are reserved. Note: This field is only valid when the Port ID field in this register (bits[19:16]) indicates the north or south internal bus memory port. Note: Not all of the Internal Bus Requesters accesses the DDR SDRAM Memory.	Internal Bus Requester ID	Requester Name	0000 ₂	Reserved	0001 ₂	Intel XScale® microarchitecture 0 (coreID0)	0010 ₂	Intel XScale® microarchitecture 1 (coreID1)	0011 ₂	ATUX	0100 ₂	ATUE	0101 ₂	Application DMAs	0110 ₂	Reserved	0111 ₂	Reserved	1000 ₂	Reserved	1001 ₂	SMBus
Internal Bus Requester ID	Requester Name																							
0000 ₂	Reserved																							
0001 ₂	Intel XScale® microarchitecture 0 (coreID0)																							
0010 ₂	Intel XScale® microarchitecture 1 (coreID1)																							
0011 ₂	ATUX																							
0100 ₂	ATUE																							
0101 ₂	Application DMAs																							
0110 ₂	Reserved																							
0111 ₂	Reserved																							
1000 ₂	Reserved																							
1001 ₂	SMBus																							



Error #		Intel XScale® microarchitecture Local Bus	Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible	
0		Address offset		
1		+1820H		
		+1824H		



7.8.11 DDR ECC Address Registers — DEAR0, DEAR1

These registers are responsible for logging the addresses where the errors were detected on the local memory bus. Two errors are detected and logged. The software knows which DDR SDRAM address had the error by reading these registers and decoding the syndrome in the log registers. The upper 4 bits of the 36-bit address are captured in the DELOGx — refer to [Section 7.8.10, DDR ECC Log Registers — DELOG0, DELOG1](#). For error details, see [Section 7.3.4, “DDR Error Correction and Detection” on page 607](#)).

Table 384. DDR ECC Address Registers – DEAR0, DEAR1

Diagram illustrating the Error Address field structure (bits 31:02). The field is divided into two main sections: IOP Attributes (bits 31:02) and PCI Attributes (bits 31:02). The IOP Attributes section is further divided into four 4-bit groups (bits 31:28, 24:20, 16:12, 8:4) and a 4-bit group (bits 3:0). The PCI Attributes section is also divided into four 4-bit groups (bits 31:28, 24:20, 16:12, 8:4) and a 4-bit group (bits 3:0). The IOP Attributes are labeled 'ro' (Read Only) for bits 31:28, 24:20, 16:12, 8:4, and 'rv' (Reserved) for bits 3:0. The PCI Attributes are labeled 'na' (Not Accessible) for all bits. The last two bits (3:02) are shaded gray.



This register is responsible for logging the address of the ADMA descriptor in bits[30:05] while an ECC error was detected on the memory bus. The DMA type is read in “DDR ECC Log Registers — DELOG0, DELOG1” on page 639. Two errors are detected and logged. The software knows which descriptor was being processed by reading this register. This register is used in conjunction with DECUARx. Refer to [Section 7.8.13, DDR ECC Context Upper Address Registers — DECUAR0, DECUAR1](#).

Error #	Intel XScale® microarchitecture Local Bus Address	Attribute Legend:
0	Offset	RV = Reserved
1	+1830H	RC = Read Clear
	+1834H	RO = Read Only
		RS = Read/Set
		NA = Not Accessible

Bit	Default	Description
31	0 ₂	Reserved.
30:05	0000000H	ADMA Error Descriptor Address: Bits[30:5] of this bit field stores the ADMA descriptor address[30:05] when an ECC error occurs. Note that ADMA descriptors are 32-byte aligned. <u>Bits</u> [30:05] = ADMA Descriptor Address[30:05].
04:00	00000 ₂	Reserved.



7.8.13 DDR ECC Context Upper Address Registers — DECUAR0, DECUAR1

This register is responsible for logging the ADMA channel number when an ECC error is detected on the memory bus. The DMA type is read in “DDR ECC Log Registers — DELOG0, DELOG1” on page 639. Two errors are detected and logged. This register is used in conjunction with DECARx. Refer to Section 7.8.12, DDR ECC Context Address Registers — DECAR0, DECAR1.

Table 386. DDR ECC Context Upper Address Registers — DECUAR 0, DECUAR 1

		31	28	24	20	16	12	8	4	0	
Error #	IOP Attributes	rv	rv	rv	rv	rv	rv	rv	rv	rv	
		rv	rv	rv	rv	rv	rv	rv	rv	rv	
0	PCI Attributes	na	na	na	na	na	na	na	na	na	
		na	na	na	na	na	na	na	na	na	
1	PCI Attributes	na	na	na	na	na	na	na	na	na	
		na	na	na	na	na	na	na	na	na	
		Intel XScale® microarchitecture Local Bus Address								Attribute Legend:	
		Offset								RW = Read/Write	
		+1838H								RV = Reserved	
		+183CH								PR = Preserved	
										RS = Read/Set	
										RC = Read Clear	
										RO = Read Only	
										NA = Not Accessible	
Bit	Default	Description									
31:04	0000 000H	Reserved									
03:02	00 ₂	DMA Channel Number: Stores the ADMA channel number when an ECC error occurs. The DMA type is read from the “DDR ECC Log Registers — DELOG0, DELOG1” on page 639.									
01:00	00 ₂	Reserved									

7.8.14 DDR ECC Test Register — DECTST

This register allows testing between the ECC logic and the memory subsystem (Section 7.3.4.6, “DDR ECC Testing” on page 616). 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.

Table 387. DDR ECC Test Register — DECTST

		31	28	24	20	16	12	8	4	0	
IOP Attributes	[	rv	rv	rv	rv	rv	rv	rv	rv	rv	
		rv	rv	rv	rv	rv	rv	rv	rv	rv	
PCI Attributes	[	na	na	na	na	na	na	na	na	na	
		na	na	na	na	na	na	na	na	na	
Intel XScale® microarchitecture Local Bus Address offset +1840H		Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set								RC = Read Clear RO = Read Only NA = Not Accessible	

Bit	Default	Description
31:08	00 0000H	Reserved
07:00	00H	ECC Mask: 8-bit ECC mask. Each ECC bit is XORed with the appropriate bit in this mask field before the ECC is stored into memory. See Section 7.3.4.6, “DDR ECC Testing” on page 616 .



7.8.15 DDR Parity Control and Status Register — DPCSR

This register programs DMCU parity checking capabilities. This register is also responsible for logging error types detected on the DDR memory ports. Only one error is detected and logged. The error recorded corresponds to the addresses in (DPAR, DPUAR) and (DPCAR, DPCUAR).

The status bits are read-only bits and only have meaning when DMCISR[8] is non-zero. For more details, see [Section 7.6, “Parity Interrupts/Error Conditions” on page 624](#).

Table 388. DDR Parity Control and Status Register — DPCSR

		Intel XScale® microarchitecture Local Bus Address offset +1844H	
		Attribute Legend: RW = Read/Write, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible	
Bit	Default	Description	
31:24	00H	Reserved	
23:20	0H	Parity Error Requester: Indicates the requester of the logged error: Internal Bus Requester ID Requester Name 0000 ₂ Reserved 0001 ₂ Intel XScale® microarchitecture 0 (coreID0) 0010 ₂ Intel XScale® microarchitecture 1 (coreID1) 0011 ₂ ATUX 0100 ₂ ATUE 0101 ₂ Application DMAs 0110 ₂ Reserved 0111 ₂ Messaging Unit 1000 ₂ Reserved 1001 ₂ SMBus All other IDs are reserved. Note: This field is only valid when the Port ID in this register (bits[19:16]) indicates the north or the south internal bus as the memory port. Note: Not all of the Requesters can or accesses the DDR SDRAM Memory.	
19:16	0H	Parity Error Direct Memory Port ID: Indicates the direct memory port of the logged error: Port ID Port Name 0000 ₂ North Internal Bus 0001 ₂ South Internal Bus 0010 ₂ Application DMA 0 0011 ₂ Application DMA 1 0100 ₂ Application DMA 2 0101 ₂ MU All other IDs are reserved.	
15:01	0000H	Reserved	
00	0 ₂	DMCU Parity Enabled: Enables or disables checking, logging, and reporting (interrupt generation) of parity errors on the memory ports. 0 = Disable Parity Error 1 = Enable Parity Error	



7.8.16 DDR Parity Address Register — DPAR

This register is responsible for logging the local memory port detected 36-bit error address. This register is used in conjunction with the [Table 390, “DDR Parity Upper Address Register — DPUAR”](#). One error is detected and logged. The software knows which DDR SDRAM address had the error by reading this register and decoding contents of associated log register. For error details, see [Section 7.3.4, “DDR Error Correction and Detection”](#) on page 607).

Table 389. DDR Parity Address Register — DPAR

		312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro rv rv PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® microarchitecture Local Bus Address Offset +1848H Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:02	0	Error Address: In 64-bit DDR memory mode this field stores lower 28 bits of the address (a 16-byte address) in bits[31:04] that resulted in a parity error. In 32-bit DDR memory mode this field stores the lower 29 bits of the address (an 8-byte address) in bits[31:03] that resulted in a parity error. Note: Bits [3:2] should be ignored in 64-bit mode and bit[2] should be ignored in 32-bit mode.																																			
01:00	00 ₂	Reserved																																			

7.8.17 DDR Parity Upper Address Register — DPUAR

This register responsible for logging DDR SDRAM memory port detected upper 4-bit (of 36-bit) error address. This register is used in conjunction with [Table 389, “DDR Parity Address Register — DPAR”](#). One error is detected and logged. The software knows which DDR SDRAM address had the error by reading this register and decoding contents of associated log register.

Table 390. DDR Parity Upper Address Register — DPUAR

31 28 24 20 16 12 8 4 0 IOP Attributes rv		



7.8.18 DDR Parity Context Address Register — DPCAR

This register is responsible for logging the address of the ADMA descriptor in bits[30:05] when a parity error is detected on the local memory bus. The DMA type is read in “[DDR Parity Control and Status Register — DPCSR](#)” on page 644. One error is detected and logged. Software determines which descriptor was being processed by reading this register. This register is used in conjunction with DPCUAR. Refer to [Section 7.8.19, DDR Parity Context Upper Address Register — DPCUAR](#).

Table 391. DDR Parity Context Address Register — DPCAR

		31	28	24	20	16	12	8	4	0
IOP Attributes	[	rv	ro	ro	ro	ro	ro	ro	ro	ro
		ro	ro	ro	ro	ro	ro	ro	ro	ro
PCI Attributes	[	na	na	na	na	na	na	na	na	na
		na	na	na	na	na	na	na	na	na
Error #	0	Intel XScale® microarchitecture Local Bus Address Offset +1850H								
		Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible								
Bit	Default	Description								
31	0 ₂	Reserved.								
30:05	0000000H	ADMA Error Descriptor Address: Bits[30:5] of this bit field stores the ADMA descriptor address[30:05] when a parity error occurs. Note that ADMA descriptors are 32-byte aligned. [30:05] = ADMA Descriptor Address[30:05].								
04:00	00000 ₂	Reserved.								

7.8.19 DDR Parity Context Upper Address Register — DPCUAR

This register is responsible for logging the ADMA channel number when an ECC error is detected on the local memory bus. The DMA type is read in “[DDR Parity Control and Status Register — DPCSR](#)” on page 644. One error is detected and logged. This register is used in conjunction with DPCAR. Refer to [Section 7.8.18, DDR Parity Context Address Register — DPCAR](#).

Table 392. DDR Parity Context Upper Address Register — DPCUAR

		31	28	24	20	16	12	8	4	0
IOP Attributes	[	rv	rv	rv	rv	rv	rv	rv	rv	rv
		rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	[	na	na	na	na	na	na	na	na	na
		na	na	na	na	na	na	na	na	na
Error #	0	Intel XScale® microarchitecture Local Bus Address Offset +1858H								
		Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible								
Bit	Default	Description								
31:04	0000 000H	Reserved								
03:02	00 ₂	DMA Channel Number: Stores the ADMA channel number when a parity error occurs. The DMA type can be read from the “ DDR Parity Control and Status Register — DPCSR ” on page 644.								
01:00	00 ₂	Reserved								

7.8.20 DDR Memory Controller Interrupt Status Register — DMCISR

Setting the DMCISR asserts an interrupt to the core. Upon an interrupt, the Intel XScale® microarchitecture polls the interrupt status register for each unit. The interrupt status register tells the core the reason for the interrupt. The DMCU has six interrupt conditions: first ECC error (DMCISR[0]), second ECC error (DMCISR[1]), more than two ECC errors (DMCISR[2]), Address Region Error (DMCISR[03]), first parity error (DMCISR[08]), and more than one parity error (DMCISR[09]).

When DDMCU detects an ECC error and both DMCISR[0] and DMCISR[1] are cleared, the error is logged in DELOG0 and DMCISR[0] is set to 1. When one DMCISR bit is not clear and DDR DMCU detects an error, the error is logged in the unused DELOGx register and the appropriate DMCISR bit is set to 1. When both DMCISR[0] and DMCISR[1] are not clear, any additional ECC errors are not logged and DMCISR[2] is set.

Bits 2:0 are read/clear bits which means that to clear them, software must write a one to these bits.

Table 393. DDR Memory Controller Interrupt Status Register – DMCISR

Intel XScale® microarchitecture Local Bus Address
offset
+1860H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible



7.8.21 DMCU Port Transaction Count Register — DMPTCR

Sets the number of transactions a given port has 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 DMCU arbiter selects a different port for DDR SDRAM transactions. This register along with the “DMCU Preemption Control Register — DMPCR” on page 650 are used to optimize the memory controller operation.

Table 394. DMCU Port Transaction Count Register — DMPTCR (Sheet 1 of 2)

Intel XScale® microarchitecture Local Bus Address offset +1868H Attribute Legend: RW = Read/Write, RC = Read Clear, RO = Read Only, NA = Not Accessible		
Bit	Default	Description
31:24	00H	Reserved
23:20	1H	MU Port Transaction Count: Number of transactions the MU port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions Note: The MU Port Transaction Count is always set to 1H.
19:16	1H	ADMA 2 Port Transaction Count: Number of transactions the ADMA 2 port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions
15:12	1H	ADMA 1 Port Transaction Count: Number of transactions the ADMA 1 port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions

**Table 394. DMCU Port Transaction Count Register — DMPTCR (Sheet 2 of 2)**

Intel XScale® microarchitecture Local Bus Address offset +1868H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
11:08	1H	ADMA 0 Port Transaction Count: Number of transactions the ADMA 0 port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions
07:04	1H	South Internal Bus Count: Number of transactions the south internal bus port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions
03:00	0H	North Internal Bus Transaction Count: Number of transactions the north internal bus port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transactions 3H = 3 transactions ... FH = 15 transactions 0H = 16 transactions



7.8.22 DMCU Preemption Control Register — DMPCR

Enables Preemption of the South Internal Bus port and the direct attached ports transaction when a North Internal Bus transaction is pending.

Table 395. DMCU Preemption Control Register - DMPCR

31 28 24 20 16 12 8 4 0 IOP Attributes rv <		

7.8.23 Frequency Register — RFR

The Refresh Frequency Register is programmed for refreshing the DDR 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 606 for recommended programmed values.

Table 396. Refresh Frequency Register — RFR

Bit	Default	Description
31:13	0	Reserved
12:00	000H	Refresh Interval: Programs number of clocks that trigger a request for a refresh cycle on DDR SDRAM interface. When all zeroes, refresh cycles are disabled. See Section 7.3.1.5, “Refresh Counter” on page 568. Note: When the memory interface is busy when the refresh counter expires, it is possible for DMCU to generate more than one refresh cycle when the memory interface becomes available.



7.8.24 DDR RCOMP Control Register — DRCR

This register provides controls for the DDR RCOMP state machine. The RCOMP state machine is used to dynamically adjust the drive strength and the slew rate in order to compensate for temperature and voltage variations. The RCOMP state machine operates using the values provided in the “DDR RCOMP Control Register — DRCR” on page 652 and “RCOMP Pad Drive Strength Select — RPDSR” on page 654.

Table 397. DDR RCOMP Control Register — DRCR (Sheet 1 of 2)

		Intel XScale® microarchitecture Local Bus offset +2000H																																			
		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:14	00000H	Reserved																																			
13	0 ₂	When this bit is set, the slew rate manual override values are used. When this bit is cleared, the slew rate is derived from the drive strength (normal operation).																																			
12:11	11 ₂	ODT Enable: Enables ODT for the DQ input pads by selecting 75 ohms or 150 ohms ODT drive strength. The RCOMP state machine uses the selected value to dynamically adjust the drive strength and the slew rate of the DQ input pads. 00 ₂ ODT is disabled 10 ₂ selects 150 ohms ODT drive strength (same as 01 ₂) 01 ₂ selects 150 ohms ODT drive strength (same as 10 ₂) 11 ₂ selects 75 ohms ODT drive strength																																			
10:09	01 ₂	15 Ohm Adjust Control: This field provides control to further adjust the value for all the 15 ohm drive strength selection made using “RCOMP Pad Drive Strength Select — RPDSR” on page 654. 00 ₂ 13.7 ohm 01 ₂ 15 ohm 10 ₂ 16.4 ohm 11 ₂ Reserved.																																			
08:07	10 ₂	21 Ohm Adjust Control: This field provides control to further adjust the value for all the 21 ohm drive strength selection made using “RCOMP Pad Drive Strength Select — RPDSR” on page 654. 00 ₂ 17.9 ohm (Recommended value for Unbuffered DIMMs). 01 ₂ 19.4 ohm 10 ₂ 21 ohm 11 ₂ Reserved.																																			
06:05	01 ₂	25 Ohm Adjust Control: This field provides control to further adjust the value for all the 25 ohm drive strength selection made using “RCOMP Pad Drive Strength Select — RPDSR” on page 654. 00 ₂ 23 ohm 01 ₂ 25 ohm 10 ₂ 27.2 ohm 11 ₂ Reserved.																																			
04:03	01 ₂	35 Ohm Adjust Control: This field provides control to further adjust the value for all the 35 ohm drive strength selection made using “RCOMP Pad Drive Strength Select — RPDSR” on page 654. 00 ₂ 32.1 ohm 01 ₂ 35 ohm 10 ₂ 38.2 ohm 11 ₂ Reserved.																																			

**Table 397. DDR RCOMP Control Register — DRCR (Sheet 2 of 2)**

		312824201612840 IOP Attributes rv																																			



This register provides the drive strength values for various pad groups. The selected drive strength values are used by the RCOMP state machine to dynamically adjust the drive strength and the slew rate in order to compensate for temperature and voltage variations. The selected values are further adjusted by using the “[DDR RCOMP Control Register — DRRCR](#)” on page 652.

Bit	Default	Description
31:12	00000H	Reserved
11:09	000 ₂	Drive Strength Select: Drive strength for CKE[1:0] and CS[1:0]# pads used by RCOMP state machine. Value can be further adjusted using "DDR RCOMP Control Register — DRCR" on page 652 . 000 ₂ 50 ohm Drive strength 001 ₂ 35 ohm Drive strength 010 ₂ 25 ohm Drive strength 011 ₂ 17.9 ohm Drive strength (Recommended value for Unbuffered DIMMs) 100 ₂ 15 ohm Drive strength All other values are reserved
08:06	000 ₂	Drive Strength Select: Drive strength for M_CK[2:0] and M_CK[2:0]# pads used by RCOMP state machine. Value can be further adjusted using "DDR RCOMP Control Register — DRCR" on page 652 . 000 ₂ 50 ohm Drive strength 001 ₂ 35 ohm Drive strength 010 ₂ 25 ohm Drive strength 011 ₂ 17.9 ohm Drive strength 100 ₂ 15 ohm Drive strength All other values are reserved
05:03	000 ₂	Drive Strength Select: Drive strength for BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] pads used by RCOMP state machine. Value can be further adjusted using "DDR RCOMP Control Register — DRCR" on page 652 . 000 ₂ 50 ohm Drive strength 001 ₂ 35 ohm Drive strength 010 ₂ 25 ohm Drive strength 011 ₂ 17.9 ohm Drive strength (Recommended value for Unbuffered DIMMs) 100 ₂ 15 ohm Drive strength All other values are reserved
02:00	000 ₂	Drive Strength Select: Select drive strength for DQ[63:0] pads used by RCOMP state machine. Value can be further adjusted using the "DDR RCOMP Control Register — DRCR" on page 652 . 000 ₂ 50 ohm Drive strength 001 ₂ 35 ohm Drive strength 010 ₂ 25 ohm Drive strength 011 ₂ 17.9 ohm Drive strength 100 ₂ 15 ohm Drive strength All other values are reserved



7.8.26 DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR

The register is used to manually control the drive strength and slew rate of the p-drivers and n-drivers for the input data bus (DQ[63:0]) signals. This register is used when the RCOMP state machine is disabled via bit 0 of the "DDR RCOMP Control Register — DRCCR" on page 652.

Note: The user may be required to program this register based on the loading of the DDR SDRAM Memory Subsystem.

Table 399. DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR

		31	28	24	20	16	12	8	4	0		
IOP Attributes		rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes		na	na	na	na	na	na	na	na	na	na	na
Intel XScale® microarchitecture Local Bus offset +2008H												
Attribute Legend:						RW = Read/Write RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible						
Bit	Default	Description										
31:12	00000H	Reserved										
11:06	100000 ₂	N-ODT drive strength manual override values for input DQ pad.										
05:00	100000 ₂	P-ODT drive strength manual override values for input DQ pad.										



7.8.27 DQ Pad Drive Strength Manual Override Values Register — DQPDSR

The register is used to manually control the drive strength and slew rate of the p-drivers and n-drivers for the data bus (DQ[63:0]) signals. This register is used when the RCOMP state machine is disabled via bit 0 of the “DDR RCOMP Control Register — DRCCR” on page 652.

Note: The user may be required to program this register based on the loading of the DDR SDRAM Memory Subsystem.

Table 400. DQ Pad Drive Strength Manual Override Values Register — DQPDSR

		31	28	24	20	16	12	8	4	0			
IOP Attributes	[	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	
		rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	rv	
PCI Attributes	[	na	na	na	na	na	na	na	na	na	na	na	
		na	na	na	na	na	na	na	na	na	na	na	
		Intel XScale® microarchitecture Local Bus offset +200CH										Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible	

Bit	Default	Description
31:22	000H	Reserved
21:15	1000000 ₂	N-drive strength: Manual override values for DQ[63:0] pad.
14:08	1000000 ₂	P-drive strength: Manual override values for DQ[63:0] pad.
07:04	1000 ₂	N-slew rate: Manual override values for DQ[63:0] pad.
03:00	1000 ₂	P-slew rate: Manual override values for DQ[63:0] pad.



7.8.28 MA Pad Drive Strength Manual Override Values Register — MAPDSR

The register is used to manually control the drive strength and slew rate of the p-drivers and n-drivers for the BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] signals. This register is used when the RCOMP state machine is disabled via bit 0 of the “DDR RCOMP Control Register — DRCCR” on page 652.

Note: The user may be required to program this register based on the loading of the DDR SDRAM Memory Subsystem.

Table 401. MA Pad Drive Strength Manual Override Values Register — ADPDSR

		IOP Attributes: 31 rv, 28 rv, 24 rv, 20 rv, 16 rv, 12 rv, 8 rv, 4 rv, 0 rv PCI Attributes: 31 na, 28 na, 24 na, 20 na, 16 na, 12 na, 8 na, 4 na, 0 na Intel XScale® microarchitecture Local Bus offset +2010H Attribute Legend: RW = Read/Write, RV = Reserved, PR = Preserved, RS = Read/Set, RC = Read Clear, RO = Read Only, NA = Not Accessible																																			
Bit	Default	Description																																			
31:22	000H	Reserved																																			
21:15	1000000 ₂	N-drive strength: Manual override values for BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] pad																																			
14:08	1000000 ₂	P-drive strength: Manual override values for BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] pad.																																			
07:04	1000 ₂	N-slew rate: Manual Override values for BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] pad.																																			
03:00	1000 ₂	P-slew rate: Manual override values for BA[2:0], MA[13:0], WE#, RAS#, CAS#, ODT[1:0] pad.																																			



7.8.29 MCLK Pad Drive Strength Manual Override Values Register — MPDSR

The register is used to manually control the drive strength and slew rate of the p-drivers and n-drivers for the M_CK[2:0] and M_CK[2:0]# signals. This register is used when the RCOMP state machine is disabled via bit 0 of the “DDR RCOMP Control Register — DRCCR” on page 652.

Note: The user is required to program this register based on the loading of the DDR SDRAM Memory Subsystem.

Table 402. MCLK Pad Drive Strength Manual Override Values Register — MPDSR

		Intel XScale® microarchitecture Local Bus offset +2014H																																			
		Attribute Legend: RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																																			
Bit	Default	Description																																			
31:22	000H	Reserved																																			
21:15	1000000 ₂	N-drive strength: Manual override values for M_CK[2:0] and M_CK[2:0]# pad.																																			
14:08	1000000 ₂	P-drive strength: Manual override values for M_CK[2:0] and M_CK[2:0]# pad.																																			
07:04	1000 ₂	N-slew rate: Manual override values for M_CK[2:0] and M_CK[2:0]# pad.																																			
03:00	1000 ₂	P-slew rate: Manual override values for M_CK[2:0] and M_CK[2:0]# pad.																																			



7.8.30 CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR

The register is used to manually control the drive strength and slew rate of the p-drivers and n-drivers for the CKE[1:0] and CS[1:0]# signals. This register is used when the RCOMP state machine is disabled via bit 0 of the "DDR RCOMP Control Register — DRCCR" on page 652.

Note: The user is required to program this register based on the loading of the DDR SDRAM Memory Subsystem.

Table 403. CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR

		31 28 24 20 16 12 8 4 0 IOP Attributes rv rv rv rv rv rv rv rv rv rv rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na																															
		Intel XScale® microarchitecture Local Bus offset +2018H																Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible															
Bit	Default	Description																															
31:22	000H	Reserved																															
21:15	1000000 ₂	N-drive strength: Manual override values for CKE[1:0] and CS[1:0]# pad.																															
14:08	1000000 ₂	P-drive strength: Manual override values CKE[1:0] and CS[1:0]# pad.																															
07:04	1000 ₂	N-slew rate Manual: override values for CKE[1:0] and CS[1:0]# pad.																															
03:00	1000 ₂	P-slew rate Manual: override values for CKE[1:0] and CS[1:0]# pad.																															



The values provided by this register is used by the DLL state machine to control the relative DLL delay between the input data bus and the input data strobe signals DQS[1:0] and DQS[1:0]#.

The master delay field plus one determines the DLL delay per element:

$$\text{Element Delay} = (\text{MCLK Period}/4) / (\text{Master Delay} + 1)$$

The slave delay field plus one determines the number of DLL delay elements:

$$DLL \text{ Delay} = \text{Element Delay} * (\text{Slave Delay} + 1)$$

Note: The Master_Delay and Slave_Delay fields in the register are encoded such that a value of 0 represents 1, a value of 1 represents 2, and so on.

Note: When necessary, the user is required to program this register to tune the data strobe delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the “[DDR SDRAM Control Register 0 – SDCR0](#)” on page 628.

Table 404. DLL Delay Register 0 — DLLR0

Bit	Default	Description
31:29	000 ₂	Reserved.
28:24	01111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS1#. Recommended value is 01101 ₂ . Note: Slave determines the number of DLL delay elements.
23:21	000 ₂	Reserved.
20:16	01111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS1. Recommended value is 01101 ₂ . Note: Slave determines the number of DLL delay elements.
15:13	000 ₂	Reserved.
12:08	01111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS0#. Recommended value is 01101 ₂ . Note: Slave determines the number of DLL delay elements.
07:05	000 ₂	Reserved.
04:00	01111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS0. Recommended value is 01101 ₂ . Note: Slave determines the number of DLL delay elements.



7.8.32 DLL Delay Register 1 — DLLR1

The values provided by this register is used by the DLL state machine to control the relative DLL delay between the input data bus and the input data strobe signals DQS[3:2] and DQS[3:2]#.

The master delay field determines the DLL delay per element:

$$\text{Element\_Delay} = (\text{MCLK\_Period} / 4) / (\text{Master\_Delay} + 1)$$

The slave delay field determines the number of DLL delay elements:

$$\text{DLL Delay} = \text{Element\_Delay} * (\text{Slave\_Delay} + 1)$$

Note: The Master_Delay and Slave_Delay fields in the register are encoded such that a value of 0 represents 1, a value of 1 represents 2, and so on.

Note: When necessary, the user is required to program this register to tune the data strobe delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the “DDR SDRAM Control Register 0 — SDCR0” on page 628.

Table 405. DLL Delay Register 1 — DLLR1

Intel XScale® microarchitecture Local Bus offset +2020H		
Attribute Legend: RW = Read/Write, RC = Read Clear, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:29	000 ₂	Reserved.
28:24	0111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS3#. Recommended value is 0110 ₂ . Note: Slave determines the number of DLL delay elements.
23:21	000 ₂	Reserved.
20:16	0111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS3. Recommended value is 0110 ₂ . Note: Slave determines the number of DLL delay elements.
15:13	000 ₂	Reserved.
12:08	0111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS2#. Recommended value is 0110 ₂ . Note: Slave determines the number of DLL delay elements.
07:05	000 ₂	Reserved.
04:00	0111 ₂	Data Strobe Slave Delay: Selects the number of slave delay elements for DQS2. Recommended value is 0110 ₂ . Note: Slave determines the number of DLL delay elements.



The values provided by this register is used by the DLL state machine to control the relative DLL delay between the input data bus and the input data strobe signals DQS[5:4] and DQS[5:4]#.

$$\text{Element Delay} = (\text{MCLK Period}/4) / (\text{Master Delay} + 1)$$
$$DLL \text{ Delay} = \text{Element Delay} * (\text{Slave Delay} + 1)$$

Note: When necessary, the user is required to program this register to tune the data strobe delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the “[DDR SDRAM Control Register 0 – SDCR0](#)” on page 628.

Table 406. DLL Delay Register 2 — DLLR2

Intel® 81341 and 81342 I/O Processors
Developer's Manual
662

December 2007
Order Number: 315037-002US



7.8.34 DLL Delay Register 3 — DLLR3

The values provided by this register is used by the DLL state machine to control the relative DLL delay between the input data bus and the input data strobe signals DQS[7:6] and DQS[7:6]#.

The master delay field determines the DLL delay per element:

$$\text{Element Delay} = (\text{MCLK Period}/4) / (\text{Master Delay} + 1)$$

The slave delay field determines the number of DLL delay elements:

$$DLL \text{ Delay} = \text{Element Delay} * (\text{Slave Delay} + 1)$$

Note: The Master_Delay and Slave_Delay fields in the register are encoded such that a value of 0 represents 1, a value of 1 represents 2, and so on.

Note: When necessary, the user is required to program this register to tune the data strobe delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the “[DDR SDRAM Control Register 0 – SDCR0](#)” on page 628.

Table 407. DLL Delay Register 3 — DLLR3

Intel XScale® microarchitecture Local Bus offset +2028H

Attribute Legend:
 RV = Read/Write
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



The values provided by this register is used by the DLL state machine to control the relative DLL delay between the input data bus and the input data strobe signals DQS[8] and DQS[8]#. This register also controls the master delay for the data strobe signals and the master DLL delay for the Receive Enable signal.

The master delay field determines the DLL delay per element:

$$\text{Element Delay} = (\text{MCLK Period}/4) / (\text{Master Delay} + 1)$$

The slave delay field determines the number of DLL delay elements:

$$DLL \text{ Delay} = \text{Element Delay} * (\text{Slave Delay} + 1)$$

Note: The Master_Delay and Slave_Delay fields in the register are encoded such that a value of 0 represents 1, a value of 1 represents 2, and so on.

Note: When necessary, the user is required to program this register to tune the data strobe delay and the receive enable delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the “[DDR SDRAM Control Register 0 — SDCR0](#)” on page 628.

Table 408. DLL Delay Register 4 — DLLR4

Intel® 81341 and 81342 I/O Processors
Developer's Manual
664

7.8.36 DLL Delay for Receive Enable Register — DLLRCVER

This register is used to select the DIMM Type, (i.e., registered versus unbuffered). This register is also used to position the receive enable signal pulse width relative to the input data strobes by programming the slave DLL delay values. Note that the receive enable master delay value is located in the “DLL Delay Register 4 – DLLR4” on [page 664](#)

Note: The Slave_Delay fields in the register are encoded such that a value of 0 represents 1, a value of 1 represents 2, and so on.

Note: When necessary, the user is required to program this register to tune the data strobe delay and the receive enable delay of the DDR SDRAM Memory Subsystem. However, after tuning any of the delay values, the Read FIFO must be reset using bit 7 of the ["DDR SDRAM Control Register 0 — SDCR0"](#) on page 628.

Table 409. DLL Delay for Receive Enable Register – DLLRCVER (Sheet 1 of 3)

Intel XScale® microarchitecture Local Bus offset +2030H

Attribute Legend:
 RV = Reserved
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible



Table 409. DLL Delay for Receive Enable Register — DLLRCVER (Sheet 2 of 3)

		31 rv 28 rv 24 rv 20 rv 16 rv 12 rv 8 rv 4 rv 0 rv IOP Attributes [rv rv rv rv rv rv rv ro rv rv rv rv rv rw rw rw rv rv rv rv rw rw rw rv rv rv rw rw rw rw rw] na

Table 409. DLL Delay for Receive Enable Register — DLLRCVER (Sheet 3 of 3)

Bit	Default	Description
04:00	01111 ₂	Receive Enable Slave Delay 0: Receive Enable Delay select lines. This bit field together with the Receive Enable Delay 1 provides an 8-bit Receive Enable Slave Delay Select bits (Bits[10:08,04:00] of this register). Recommended values are provided below for board trace impedance of 50 and 60 Ohms: For DDR2 400MHz / 50 Ohms: Trace Length (Inches) Receive Enable Slave Delay 2 00110 ₂ 4 01101 ₂ 6 10110 ₂ 8 11110 ₂ For DDR2 400MHz / 60 Ohms: Trace Length (Inches) Receive Enable Slave Delay 2 00101 ₂ 4 01101 ₂ 6 10101 ₂ 8 11110 ₂ For DDR2 533MHz / 50 Ohms: Trace Length (Inches) Receive Enable Slave Delay 2 00011 ₂ 4 01101 ₂ 6 11001 ₂ 8 00100 ₂ For DDR2 533MHz / 60 Ohms: Trace Length (Inches) Receive Enable Slave Delay 2 00010 ₂ 4 01101 ₂ 2 11000 ₂ 2 00011 ₂

7.8.36.1 Determining DLLRCVEREN Pulse Optimum Location

The pseudo-code listed below provides an algorithm to determine the optimum location for the DLLRCVEREN pulse with respect to the DQS signals. The rising and falling edges of the DLLRCVEREN pulse must be placed in the pre-amble and post-ambly windows driven by the DDR2 memory as shown in Figure 99. The DLLRCVEREN pulse has a fixed width. The entire pulse is shifted by controlling the DLLRCVEREN delay value. The basic goal of the algorithm is to locate the first rising edge of the DQS signal and use the result to calculate the optimum DLLRCVEREN delay value. Note that the SDRAM initialization sequence must be executed before executing this algorithm. Refer to Section 7.3.3.7, “DDR SDRAM Initialization” on page 594.

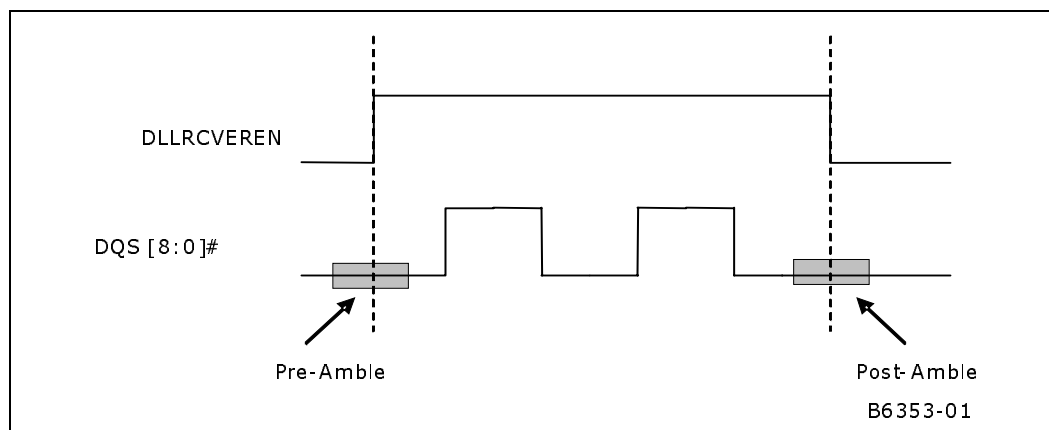
The algorithm starts by taking an initial sample of the DQS signal value using a given DLLRCVEREN delay value. The DQS sample indicates the value of the DQS signal at the leading edge of the DLLRCVEREN pulse as shown in Figure 99. Using the initial DQS value, the algorithm proceeds to find the rising edge of DQS. The DQS signal rising edge is searched by shifting the DLLRCVEREN pulse either left or right depending on the first sampled DQS value. For example, when the first sampled DQS value is '1', DLLRCVEREN delay is decreased one delay element at a time until the DQS rising edge is located. And when the first sampled DQS value is '0', the DLLRCVEREN delay is increased one delay element at a time until the rising edge of DQS is located.

Once a DQS rising edge is located, the algorithm proceeds to verify that this is the first DQS rising edge. For example, the DQS rising edge found may be the second rising edge. When it is determined that it is indeed the second DQS rising edge, the DLLRCVEREN delay is adjusted by subtracting one full MCLK worth of delay from the current DLLRCVEREN delay value.

At this stage in the algorithm, with the current DLLRCVEREN delay value, the DLLRCVEREN leading edge must be aligned with the first rising edge of the DQS signal. To calculate the optimum DLLRCVEREN delay value, MCLK/4 worth of delay must be subtracted from the current DLLRCVEREN delay value.

After programming the correct DLLRCVEREN delay value, the FIFO must be reset. SDCR0 bit 07.

Figure 99. DLLRCVREN Optimum Pulse Location





7.8.36.1.1 DLLRCVREN Algorithm

```

#define MCU_DLLRCVER      ((volatile unsigned int *) 0xffd82030)
#define MCU_SDCR0        ((volatile unsigned int *) 0xffd81804)

int compute_dllrcver(void)
{
    int dllrcver_value, dqs_value, treg, fine, coarse;
    int inverted_dqs_value, direction_value;
    unsigned int sdcr0_val;

    *MCU_DLLRCVER &= ~(1<<18); /* Clear bit 18 to clear Result bit */

    dllrcver_value = 0x50; /* Set Initial Value of DLLRCVER Delay using a
                           recommended initial value */

    dqs_value = sample_dqs(dllrcver_value); /* Take first sample of DQS with
                                           current DLLRCVER Delay */

    /* Set direction and compare value to find next edge */
    inverted_dqs_value = (dqs_value == 0) ? 1 : 0;
    direction_value = (dqs_value == 0) ? +1 : -1; /* look for rising edge */

    do {
        dllrcver_value += direction_value;
        dqs_value = sample_dqs(dllrcver_value);
        if (dqs_value <= 0 || dqs_value >= 255)
            return -1; /* something went wrong */
    } while (dqs_value != inverted_dqs_value);

    /* Verify if the DQS edge found above is the second DQS edge, and
       adjust DLLRCVER Value if it was indeed the second edge */

    if (dllrcver_value >= 0x30 && sample_dqs(dllrcver_value - 0x30))
        dllrcver_value -= 0x40; /* Shift back to first edge

```



```
/* Compute Optimum DLLRCVER Value by moving DLLRCVEREN delay back
   by MCLK/4 to be in the pre-ample window */

dllrcver_value -= 16;

/* Make sure we don't go below zero */
if(dllrcver_value < 0)
    dllrcver_value = 0;

/* Write new DLLRCVER value */
treg = *MCU_DLLRCVER & (1<<16);
fine = (dllrcver_value & 0x1f);
coarse = ((dllrcver_value >> 5) & 0x7);
*MCU_DLLRCVER = ( treg | (coarse << 8) | fine );

/* Reset the MCU FIFOs */
sdcr0_val = *(MCU_SDCR0);
*(MCU_SDCR0) = sdcr0_val &~ (1<<7); /* Clear Reset FIFO bit */
*(MCU_SDCR0) = sdcr0_val | (1<<7); /* Set Reset FIFO bit */

return 0;

} // End Algorithm

/* Sample DQS Procedure */

sample_dqs(dllval)
{
    unsigned int treg, fine, coarse, val;
    volatile unsigned int *ddr_mem_addr = 0; /* Pick a DDR addr */

    treg = *MCU_DLLRCVER & (1<<16);
    fine = (dllval & 0x1f);
    coarse = ((dllval >> 5) & 0x7);
```



```
/* Write DLLRCVER Register */
*MCU_DLLRCVER = ((1<<17) | (1<<18) | treg | (coarse << 8) | fine);

/* Read register again to guarantee previous write operation */
*MCU_DLLRCVER;

/* Read a DDR SDRAM Memory Location, this memory read causes
   the auto calibration circuit to sample the DQS signal */
val = *ddr_mem_addr;

/* If not defined as a strongly ordered memory region, may need to
   put a dependancy here to ensure load from DDR completes */
asm volatile("mov %0, %0" : : "r" (val));

/* Read DLLRCVER register bit 24 to get DQS sample */
val = *MCU_DLLRCVER & (1<<24);
return (val == 0 ? val : 1);

} // End sample_dqs() procedure
```



7.8.37 DDR RCOMP Status Register for 15 Ohm RCOMP — DRSR15

The register is used by software to read the drive strength values computed by the RCOMP state machine for 15 Ohm RCOMP.

Table 410. DDR RCOMP Status Register for 15 Ohm RCOMP — DRSR15

		31	28	24	20	16	12	8	4	0	
IOP Attributes		rv	rv	rv	rv	rv	rv	rv	rv	rv	
PCI Attributes		na	na	na	na	na	na	na	na	na	
Intel XScale® microarchitecture Local Bus offset +2034H		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible									

Bit	Default	Description
31:22	000H	Reserved
21:15	xH	N-drive strength: Drive Strength computed by RCOMP for 15 Ohm RCOMP.
14:08	xH	P-drive strength: Drive Strength computed by RCOMP for 15 Ohm RCOMP.
07:04	0H	Reserved.
03:00	0H	Reserved.

7.8.38 DDR RCOMP Status Register for 17.9 Ohm RCOMP — DRSR17

The register is used by software to read the drive strength and slew rate values computed by the RCOMP state machine for 17.9 Ohm RCOMP.

Table 411. DDR RCOMP Status Register for 17.9 Ohm RCOMP — DRSR17

		31	28	24	20	16	12	8	4	0	
IOP Attributes		rv	rv	rv	rv	rv	rv	rv	rv	rv	
PCI Attributes		na	na	na	na	na	na	na	na	na	
Intel XScale® microarchitecture Local Bus offset +2038H		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible									

Bit	Default	Description
31:22	000H	Reserved
21:15	xH	N-drive strength: Drive Strength computed by RCOMP for 17.9 Ohm RCOMP.
14:08	xH	P-drive strength: Drive Strength computed by RCOMP for 17.9 Ohm RCOMP.
07:04	0H	Reserved.
03:00	0H	Reserved.



7.8.39 DDR RCOMP Status Register for 25 Ohm RCOMP — DRSR25

The register is used by software to read the drive strength and slew rate values computed by the RCOMP state machine for 25 Ohm RCOMP.

Table 412. DDR RCOMP Status Register for 25 Ohm RCOMP — DRSR25

Bit	Default	Description
31:22	000H	Reserved
21:15	xH	N-drive strength: Drive Strength computed by RCOMP for 25 Ohm RCOMP.
14:08	xH	P-drive strength: Drive Strength computed by RCOMP for 25 Ohm RCOMP.
07:04	0H	Reserved.
03:00	0H	Reserved.

7.8.40 DDR RCOMP Status Register for 35 Ohm RCOMP — DRSR35

The register is used by software to read the drive strength and slew rate values computed by the RCOMP state machine for 35 Ohm RCOMP.

Table 413. DDR RCOMP Status Register for 35 Ohm RCOMP — DRSR35

Bit	Default	Description
31:22	000H	Reserved
21:15	xH	N-drive strength: Drive Strength computed by RCOMP for 35 Ohm RCOMP.
14:08	xH	P-drive strength: Drive Strength computed by RCOMP for 35 Ohm RCOMP.
07:04	0H	Reserved.
03:00	0H	Reserved.



The register is used by software to read the drive strength and slew rate values computed by the RCOMP state machine for 50 Ohm RCOMP.

Intel XScale® microarchitecture Local Bus offset +2044H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 PR = Preserved
 RO = Read Only
 RS = Read/Set
 NA = Not Accessible



8.0 SRAM Memory Controller

This chapter describes the integrated SRAM Memory Controller Unit (SMCU). The operating modes, initialization, and implementation are detailed in this chapter.

8.1 Overview

The Intel® 81341 and 81342 I/O Processors (81341 and 81342) integrates a high performance, multi-ported SRAM Memory Controller to provide access to the on-chip 1.0 MByte SRAM Memory. The SRAM Memory Controller supports:

- 1.0 MByte SRAM Memory
- Dedicated port for Intel XScale® processor to the SRAM
- Optimized core processor data processing 32-bit region.
- Single-bit error correction, multi-bit detection support (ECC)
- 256-bit wide SRAM Memory Interface
- 128-bit wide port with data parity protection
- ECC supported on 32-bit data width

The SRAM interface provides a direct connection to a high bandwidth and reliable memory subsystem. An 7-bit Error Correction Code (ECC) across every 32-bit word improves system reliability. The ECC is stored into the SRAM array along with the 32-bit data and is checked when the data is read. If the code is incorrect, the SMCU 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 SMCU responds to the north internal bus, memory accesses within its programmed address range and issues the memory request to the SRAM interface.
- The SMCU contains transaction queues for the north internal bus port enabling pipelining of transactions to the SRAM for maximum performance.
- ECC implemented on 32-bit data for higher core write performance core by avoiding Read-Modify-Write (RMW) operation to the SRAM.



8.2 Glossary

This section lists commonly used terms throughout this chapter:

Table 415. Commonly Used Terms

Term	Definition
Scrubbing	Once an error is detected within the memory array, the SMCU must correct the error (if possible) while delivering the data to the initiator. Correcting the memory location is referred to as “scrubbing the array.” The SMCU relies on software to scrub any errors.
Syndrome	A syndrome is a value which indicates an error in the data read from the memory array. The SMCU computes the syndrome with every memory read. Decoding the syndrome indicates: the bit in error for a single-bit error, or a multi-bit error. Table 416 defines the syndrome decoding.

8.3 Theory of Operation

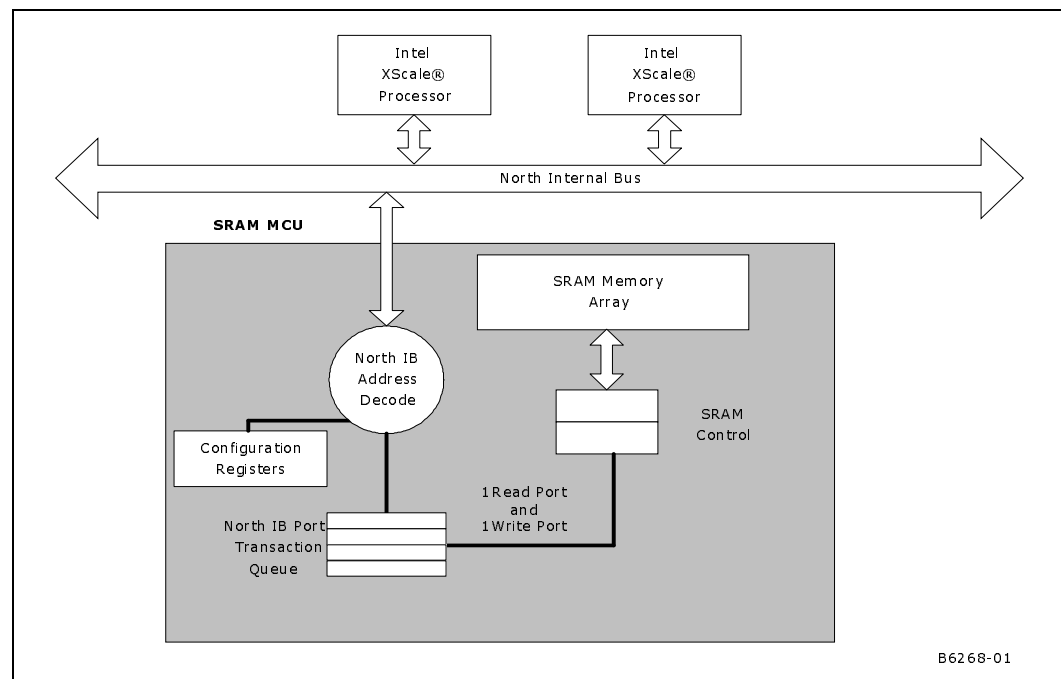
The 81341 and 81342 SRAM memory controller translates transactions from the north internal bus into the protocol supported by the SRAM memory subsystem.

8.3.1 Functional Block

The SRAM memory controller logically comprises the blocks illustrated in Figure 100. The SMCU supports a separate read and write port.

8.3.1.1 Transaction Ports

Figure 100. Intel® 81341 and 81342 I/O Processors SRAM Memory Controller Block Diagram



The SMCU provides two ports for direct SRAM access. Each device connects to the SMCU using a separate read port and write port. Each port provides a 128-bit data path. The ports are described in the next sub-sections:

8.3.1.1.1 North Internal Bus Ports

The North Internal Bus Port provides the 81341 and 81342 processor core access to the SRAM Memory Controller. This North Internal Bus Port allows core transactions targeting the SRAM via the North Internal Bus bus to pass directly to the SRAM.



8.3.1.2 Address Decode Blocks

Address Decode is performed for transactions from input North Internal Bus port to determine if the SMCU should claim the transaction. The north internal bus port claims transactions targeting the SRAM memory array space, and transactions targeting the memory-mapped registers.

8.3.1.2.1 SRAM Memory Array Space

The SRAM Memory Array Space can be accessed from the north internal bus port and the SRAM memory space is defined with the SRAM Base Address Registers (SBAR and SBUAR).

8.3.1.2.2 Memory-Mapped Register Space

The SMCU PMMR memory space offset is +1500H to +157FH. The registers are detailed in [Section 8.6, “Register Definitions” on page 699](#).

The Memory-mapped registers are only accessible from the north internal bus port.

8.3.1.2.3 North Internal Bus Port Address Decode

North internal bus transactions are decoded to determine if they address the SRAM Memory space or the memory-mapped registers. If the transaction addresses the SRAM Memory Space, the transaction is queued in the north internal bus port transaction queue.

8.3.1.3 Memory Transaction Queues

There are one set of transaction queues for transactions which address the SRAM Memory Space from the north internal bus. The transaction queues are located in each respective unit.

8.3.1.3.1 North Internal Bus Port Transaction Queue (NIBPTQ)

The NIBPTQ stores memory transactions from the north internal bus. The NIBPTQ supports 16 read transactions, each with up to 32 bytes buffer. The NIBPTQ also supports 16 posted write transactions up to 32 bytes each.

8.3.1.4 Configuration Registers

The Configuration Registers block contains all of the memory-mapped registers listed in [Section 8.6, “Register Definitions” on page 699](#). These registers define the memory subsystem connected to the 81341 and 81342. The status registers indicate the current SMCU status.

8.3.1.5 SRAM Control Block

The SRAM Control Block contains all functionality to process the SRAM data accesses per the transactions issued by the SMARB. To process a transaction the SRAM Control Block employs several sub-blocks. The sub-blocks include the SRAM State Machine and Pipeline Queues, and Error Correction Logic.

8.3.1.5.1 SRAM State Machine and Pipeline Queues

Since the SMCU generates error correction codes based on the data, the SMCU is a pipelined architecture. Pipelining also ensures acceptable AC timings to the memory interfaces. The SRAM state machine pipelines SRAM memory operations for several clocks.



8.3.1.5.2 Error Correction Logic

The Error Correction Logic generates the ECC code for SRAM 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 8.3.3, "Error Correction and Detection" on page 683](#) for more details.



8.3.1.6 North Internal Bus Port Transaction Ordering

Since requests from the north internal bus port are queued in the NIBPTQ, the port needs to maintain order of requests addressing the SRAM. Coherency between the north internal bus port and the other ports are maintained by the SMCU as described in “SMCU Port Coherency” below.

8.3.1.7 SMCU Port Coherency

With the queueing of SRAM transactions in multiple ports, coherency of memory must be maintained. The SMARB maintains memory coherency by ensuring that all writes to a given memory address are completed before any read to the same address is processed. This address comparison is done with a 1 KByte granularity.

The current read transaction is compared to all pending write transactions in the NIBPTQ transaction queue. If a write transaction is pending for the same memory location (1 KByte granularity), the write is allowed to complete first, before the read transaction can be processed. Also, to maintain ordering rules, all write transactions preceding the ‘incoherent write’ are also processed.



8.3.2 SRAM Memory Interface Support

The 81341 and 81342 memory controller supports 1.0 Mbytes of on-chip SRAM. The SMCU supports a 256-bit data bus width memory with ECC. The SMCU supports 7-bit ECC on every 32-bit data quantity, providing higher performance when the core processor is processing data by eliminating any RMW cycle required for 4-Byte store ECC generation.

The SMCU supports seamless read/write bursting of long data streams.

8.3.2.1 SRAM Initialization

Initialization software should initialize the entire SRAM memory array in order to have the correct ECC values for each ECC location. Refer to [Section 8.3.3, "Error Correction and Detection" on page 683](#)) for more details. Reading from an uninitialized memory location will result in an ECC error. By default data parity checking is disabled, firmware must enable data parity checking if required. Refer to [Section 427, "SRAM Parity Control and Status Register — SPARCSR" on page 706](#).

8.3.2.2 SRAM Read Sequence

Read transactions require ECC codes to be calculated and compared with the ECC returned by the SRAM array. The following steps describe the read sequence.

1. Each of the SMCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
 - If the address falls in the SRAM address range indicated by the SRAMBAR and SRAMUBAR the SMCU claims the transaction.
2. Once the SMARB selects the highest priority port transaction, it forwards the transaction to the SRAM control block.
3. Upon receipt of the data, the SRAM Control Block calculates the ECC code from the data and compares it with the ECC returned by the SRAM array. [Section 8.3.3, "Error Correction and Detection" on page 683](#) explains the ECC algorithm in more detail.
4. Assuming the calculated ECC matches the read ECC, the SRAM Control Block drives the data back to the corresponding memory port.
 - For each burst read issued, the memory controller increments the address by sixteen.

The SMCU continues to return data to the corresponding memory port based on the byte count of the transaction.



8.3.2.3 SRAM Write Sequence

Write transactions require ECC codes to be generated and stored in the SRAM array with the data being written. The behavior is different depending on the size of the data being written. [Section 8.3.3, "Error Correction and Detection" on page 683](#) explains the ECC algorithm in more detail.

1. Each of the SMCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
 - If the address falls in the SRAM address range indicated by the SRAMBAR and SRAMUBAR the SMCU claims the transaction.
2. Once the SMARB selects the highest priority port transaction, it forwards the transaction to the SRAM control block.
 - The ECC logic generates the ECC code for the data to be written.
 - The SRAM Control Block drives the new data to the memory array each cycle until the transaction is completed with the byte count expiring.
 - For each burst issued, the SRAM Control Block increments the address by sixteen.
 - When the data to write is less than an aligned DWORD, the SRAM Control Block will perform a read-modify-write of the entire 4 byte aligned DWORD and incorporate the new data while regenerating ECC.



8.3.3 Error Correction and Detection

The SMCU is capable of correcting any single bit errors and detecting any double bit errors in the 81341 and 81342 SRAM memory subsystem. ECC enhances the reliability of a memory subsystem by correcting single bit errors caused by electrical noise or occasional alpha particle hits on the SRAM memory array.

Similar to parity, which simply detects single bit errors, error correction requires an additional 7-bit code word for the 32-bit datum. The 81341 and 81342 implements a 256-bit data path to the SRAM array, but a 7-bit error correction code per every 32-bit datum. For example, **SCB0[6:0]** for **DQ[31:0]**, **SCB1[6:0]** for **DQ[63:32]**, **SCB2[6:0]** for **DQ[95:64]** and so on, resulting in a 312-bit wide memory subsystem. During SRAM read cycles, the SRAM Control Block detects single bit errors and corrects the data prior to returning the data to the respective memory transaction queue. SRAM write cycles generate the ECC and sends it with the data to the memories.

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 81341 and 81342, scrubbing is handled by software. If error reporting is enabled, the SMCU logs the error type in SELOG and the address in SECAR and SECUAR when an error occurs.

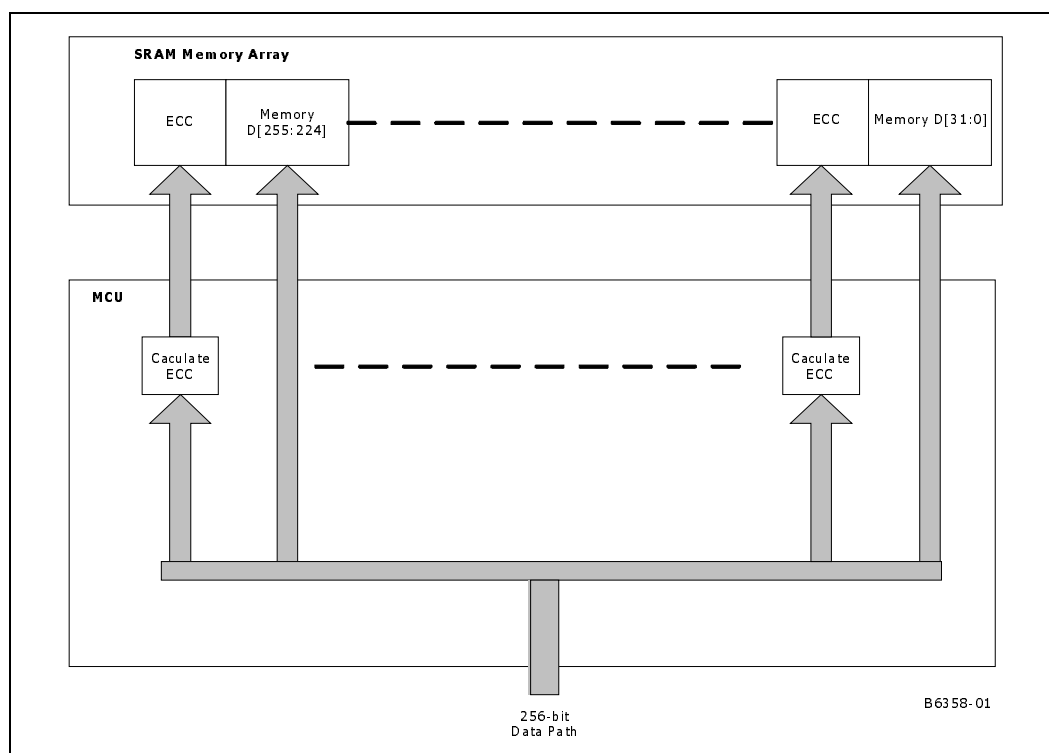
8.3.3.1 ECC Generation

For write operations, the SMCU generates the error correction code which is written along with the data. This section describes the operation of the SRAM Control Block for ECC generation on 32-bit data of the 256-bit wide memory. The SMCU will generate 7-bit wide ECC on every 32-bit data. The algorithm for a write transaction is:

```
if data to write is 32-bit wide
    Generate the ECC_with the G-matrix
    Write the new data and ECC
else {Partial Write}
    Read entire 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 101 shows how the data logically flows through the ECC hardware for a write transaction.

Figure 101. ECC Write Flow



The G-Matrix in Figure 101 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' codes of **DQ[31:0]**, **DQ[63:32]**, **DQ[95:64]**, or **DQ[255:224]** needs to be XORed together to form the ECC code. The resulting ECC bits are driven on **SCB0[6:0]**, **SCB1[6:0]**, **SCB2[6:0]**, and **SCB7[6:0]** respectively.



8.3.3.2 ECC Generation for Partial Writes

Figure 102. 81341 and 81342 G-Matrix (generates the ECC)

Data Bit	ECC Code	ECC Check Bits (E[6:0])								Data Bit	ECC Code	ECC Check Bits (E[6:0])							
		E 6	E 5	E 4	E 3	E 2	E 1	E 0	E 6			E 5	E 4	E 3	E 2	E 1	E 0		
D31	2CH		X		X	X			D15	45H	X				X		X		
D30	4AH	X			X		X		D14	51H	X		X				X		
D29	1AH			X	X		X		D13	43H	X				X	X			
D28	29H		X		X			X	D12	61H	X	X					X		
D27	5EH	X		X	X	X	X		D11	25H		X			X		X		
D26	6BH	X	X		X		X	X	D10	31H		X	X				X		
D25	2AH		X		X		X		D9	13H			X			X	X		
D24	3BH		X	X	X		X	X	D8	5BH	X		X	X		X	X		
D23	64H	X	X			X			D7	46H	X				X	X			
D22	26H		X			X	X		D6	32H		X	X			X			
D21	3EH		X	X	X	X	X		D5	23H		X				X	X		
D20	15H			X		X		X	D4	68H	X	X		X					
D19	34H		X	X		X			D3	4CH	X			X	X				
D18	54H	X		X		X			D2	52H	X		X			X			
D17	37H		X	X		X	X	X	D1	62H	X	X				X			
D16	6EH	X	X		X	X	X		D0	49H	X			X			X		

If the memory transaction writes less than 32-bit data, then the SRAM Control Block translates the write transaction into a read-modify-write transaction. For a partial write, the SRAM Control Block calculates the ECC for the modified datum and writes it back. So, if an external unit issues a write cycle with partial data to an SMCU port, the SMCU:

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 SMCU 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 SMCU causes an interrupt to the core by writing to the MCISR. The memory location is overwritten by the SMCU with the error data but valid ECC, making the contents of memory invalid. For more details on how the SMCU handles error conditions, see [Section 8.4, "ECC Interrupts/Error Conditions" on page 695](#).



8.3.3.3 ECC Checking

The ECC logic uses the following ECC read algorithm. This algorithm corrects the data before it's driven onto the internal bus. The ECC algorithm for a read transaction is:

```
Read 32-bit data and 7-bit ECC
```

```
Compute the syndrome by passing the 32-bit data through the G-Matrix and XORing the 7-bit result with the 7-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 SMCU reads the ECC from the memory subsystem, it is compared (XORed) with an ECC result that the SMCU generates from the data read from the memory. The resulting value of the XOR operation is called the syndrome. [Table 416](#) shows how the SMCU decodes the syndrome for SRAM read cycles.

Table 416. Syndrome Decoding

Error Type	Symptom
None	The syndrome is 0000 0000.
Single-Bit	Use the H-Matrix in Figure 104 to determine which bit the SMCU will invert to fix the error.
Multi-Bit	If the Syndrome does not match an 7-bit value in the H-matrix, the error is uncorrectable



Figure 103 shows how the data flows through the ECC hardware for a read transaction.

Figure 103. ECC Read Data Flow

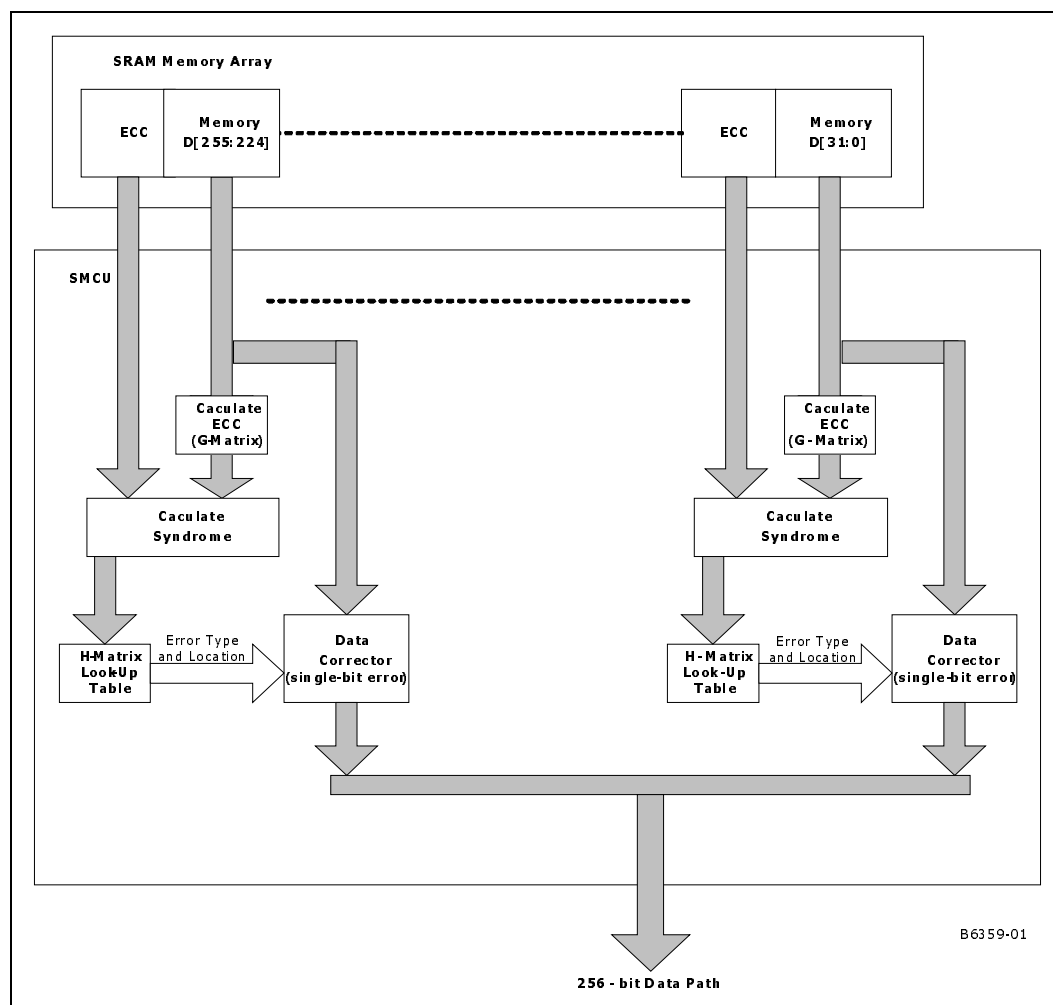




Figure 104 illustrates the H-Matrix used for decoding the syndrome. For single-bit errors, the H-Matrix indicates the bit that contains the error and consequently, which bit to fix.

Figure 104. 81341 and 81342 H-Matrix (indicates the single-bit error location)

Data Bit	Syndrome	ECC Check Bits (E[6:0])							Data Bit	Syndrome	ECC Check Bits (E[6:0])						
		E 6	E 5	E 4	E 3	E 2	E 1	E 0			E 6	E 5	E 4	E 3	E 2	E 1	E 0
E6	40H	X							D15	45H	X				X		X
E5	20H		X						D14	51H	X		X				X
E4	10H			X					D13	43H	X					X	X
E3	08H				X				D12	61H	X	X					X
E2	04H					X			D11	25H		X			X		X
E1	02H						X		D10	31H		X	X				X
E0	01H							X	D9	13H			X			X	X
D31	2CH		X		X	X			D8	5BH	X		X	X		X	X
D30	4AH	X			X		X		D7	46H	X				X	X	
D29	1AH			X	X		X		D6	32H		X	X			X	
D28	29H		X		X			X	D5	23H		X				X	X
D27	5EH	X		X	X	X	X		D4	68H	X	X		X			
D26	6BH	X	X		X		X	X	D3	4CH	X			X	X		
D25	2AH		X		X		X		D2	52H	X		X			X	
D24	3BH		X	X	X		X	X	D1	62H	X	X				X	
D23	64H	X	X			X			D0	49H	X			X			X
D22	26H		X			X	X										
D21	3EH		X	X	X	X	X										
D20	15H			X		X		X									
D19	34H		X	X		X											
D18	54H	X		X		X											
D17	37H		X	X		X	X	X									
D16	6EH	X	X		X	X	X										
D20	15H			X		X		X	D4	68H	X	X		X		X	



Referring to [Figure 103](#), the syndrome bits are created by XORing the ECC code bits as indicated by the appropriate row of the G-Matrix in [Figure 102](#) with the corresponding ECC bits read from memory. For example, the SMCU derives syndrome bit 0 by XORing ECC code associated with data bits 0, 2, 5, 8...15, 17, 20, 22, 24, 26, 28 and ECC bit 0 (physically read on **SCB[0]**). The SMCU performs seven such XOR operations (one per syndrome bit).

If decoding the syndrome indicates multi-bit error (see [Table 416](#)), the transaction results in a target-abort 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 ECC Error reporting is enabled, the SMCU 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 104](#)). For example, if the syndrome was 100 1001, the error is with bit 0 of **DQ[31:0]**. The SMCU inverts bit 0 before driving the data on internal bus.

If error reporting is enabled in the SECCR and the SMCU detects a single-bit or multi-bit error, the SMCU stores the address in SECAR and the syndrome in SELOG. Then, the SMCU signals an interrupt to the core. Software decides how to proceed through an interrupt handler. By registering the address in SECAR, software can identify the faulty location.

For details about the SMCU error conditions and how the MMR registers are affected, refer to [Section 8.4, "ECC Interrupts/Error Conditions" on page 695](#).

8.3.3.4 Scrubbing

Fixing the data error in memory is called scrubbing. The 81341 and 81342 relies on Intel XScale® processor software to perform the scrubbing. When the SMCU detects an error during a read, the SMCU logs the address where the error occurred and interrupts the core. The core 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 SMCU 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 the 32-bit data using a `ld` instruction and write the data back with a `st` instruction. Software should isolate activity on the memory location to guarantee animosity.

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 (`SECCR[0]`) during the scrubbing routine. Multi-bit errors cannot be fixed by the H-Matrix.

8.3.3.4.1 ECC Example Using the H-Matrix

Assume the core writes 9ABC DEF0H to the SRAM memory space on **DQ[31:0]**.

Using the G-Matrix in [Figure 102](#), the SRAM Control Block creates each check bit by XORing the appropriate bits in the row. Using 9ABC DEF0H, the ECC code generated is 11H. This code is written with the data to the SRAM memory on **SCB[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 core issues a read transaction to the SRAM memory. Upon the receipt of 9ABE DEF0H on **DQ[31:0]**, the SRAM Control Block calculates the syndrome with the G-Matrix in [Figure 102](#). The SRAM Control Block calculates a syndrome of 37H.

Note: During a memory write, ECC code is created by XORing the appropriate data bits' ECC codes indicated by the G-Matrix. The syndrome is created during a memory read by XORing the 7-bit value generated by XORing appropriate data bits' ECC codes indicated by the G-Matrix with the check bits (**SCB[7:0]**).

Referring to [Table 416](#), 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 37H matches a value in the H-Matrix (see [Figure 104](#)) which indicates that bit 17 has an error. The SRAM Control Block inverts bit 17 prior to returning the corrected data on the internal bus. The SMCU returns 9ABC DEF0H on the internal bus.

Assuming this was the first error the SMCU records the address where the error occurred in `SECAR` and error type in `SELOG`. If error reporting is enabled in the `SECCR`, the SMCU writes a 1 to `SMCISR[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.



8.3.3.5 ECC Disabled

If software disables ECC, the SMCU does generate the ECC byte for writes, but does not check the ECC byte for reads.

8.3.3.6 ECC Testing

[Section 8.3.3.4, “Scrubbing” on page 690](#) explains how software is responsible for correcting an error in the memory array once it has been detected by the ECC logic. The SMCU implements the SECTST register providing the programmer the ability to test error handling software. For write transactions, the SECTST 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 SMCU reads the address later, the ECC mismatches and the error condition occurs (see [Section 8.4, “ECC Interrupts/Error Conditions” on page 695](#)).

8.3.4 Byte Parity Checking and Generation

All the direct memory ports of the SMCU supports byte-wise parity on the data bus.

For write requests made to an SMCU port, the direct memory port interface performs the following tasks when receiving data:

- checks for data parity on the incoming data (parity error is logged if detected)
- generates ECC on the data

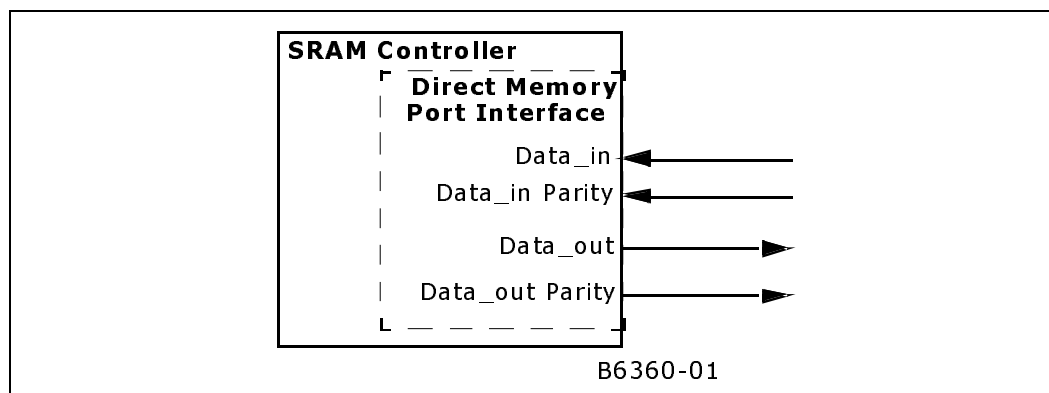
For the memory ports that are connected to the south internal bus, the write data parity is received from the south internal bus as driven by the initiator of the data on the south internal bus. For example, south internal bus initiators generate data parity. Since the north internal bus does not support parity, data parity is generated by the north internal bus port as the data enters the port and before it is written to the data queue.

For read requests made to an SMCU port, the direct memory port interface performs the following tasks before delivering data:

- checks for ECC on the data read from memory
- generates data parity on the data before delivering it on the port

For the memory ports that are connected to the south internal bus, the read data parity that are generated by the SMCU are directly driven onto the south internal bus to the requester. The data parity is then verified by the requester. Since the north internal bus does not support parity, the read data parity that are generated by the SMCU are verified by the north internal bus port when the data is read out of the data queue and before being delivered onto the north internal bus. If the north internal bus port detects a parity error while reading the data out of the data queue, the north internal bus port will return a target abort.

Figure 105. Logical Data Access Paths with Parity Protection





8.3.4.1 Parity Generation

The direct memory port interface of the SMCU only generates data parity before delivering data onto the port. After the requested data is read from memory and ECC has been verified, the direct memory port interface generates even data parity before delivering the data on the direct memory port. Table 417 lists the data bytes that are used for data parity calculation. The parity bits are calculated by bit XORing the data bits as shown in Table 417. As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is calculated as follows:

Note: The direct memory port does not support address parity.

Equation 32. $D\_PARITY0 = D[0] \text{ XOR } D[1] \text{ XOR } D[2] \text{ XOR } D[3] \text{ XOR } D[4] \text{ XOR } D[5] \text{ XOR } D[6] \text{ XOR } D[7] \text{ XOR } BE[0]$

Table 417. Data Parity Checking/Generation

Data Parity Bit	Data Byte	Byte Enable
D_PARITY15	D[127:120]	BE[15]
D_PARITY14	D[119:112]	BE[14]
D_PARITY13	D[111:104]	BE[13]
D_PARITY12	D[103:96]	BE[12]
D_PARITY11	D[95:88]	BE[11]
D_PARITY10	D[87:80]	BE[10]
D_PARITY9	D[79:72]	BE[9]
D_PARITY8	D[71:64]	BE[8]
D_PARITY7	D[63:56]	BE[7]
D_PARITY6	D[55:48]	BE[6]
D_PARITY5	D[47:40]	BE[5]
D_PARITY4	D[39:32]	BE[4]
D_PARITY3	D[31:24]	BE[3]
D_PARITY2	D[23:16]	BE[2]
D_PARITY1	D[15:8]	BE[1]
D_PARITY0	D[7:0]	BE[0]



8.3.4.2 Parity Checking

The direct memory port interface of the SMCU only checks data parity while receiving data. The direct memory port interface verifies data parity on the port interface, and then generates ECC before writing the data to memory. [Table 417](#) lists the data bits that are used for the parity calculation. The parity bits are calculated by bit XORing the data bits as shown in [Table 417](#). As an example, the parity calculation for the lowest order byte of the data bus D[7:0] is calculated as follows:

Note: The direct memory port does not support address parity.

Equation 33. $\text{DATA\_PARITY\_RESULT} = \text{D\_PARITY0} \text{ XOR } \text{D}[0] \text{ XOR } \text{D}[1] \text{ XOR } \text{D}[2] \text{ XOR } \text{D}[3] \text{ XOR } \text{D}[4] \text{ XOR } \text{D}[5] \text{ XOR } \text{D}[6] \text{ XOR } \text{D}[7] \text{ XOR } \text{BE}[0]$

A non-zero result from the above operation indicates a parity error.

The parity logic uses the following algorithm, and this algorithm logs the error if an error is detected.

```
check data parity
    if data parity result is good
        done
    else {error}
        create an error log
        Interrupt the core (if enabled)
```

8.3.4.3 Parity Disabled

If software disables parity, the SMCU would generate data parity as explained above, but would not check and report data parity on the interface.

8.3.4.4 Parity Testing

The System Controller provides the ability for the programmer to test error handling software by forcing address or data parity error. Refer to the [Chapter 6.0, "System Controller \(SC\) and Internal Bus Bridge"](#) for more details.



8.4 ECC Interrupts/Error Conditions

The SMCU has two ECC conditions which require intervention from the Intel XScale® processor. If a single-bit error is detected during a read cycle, the SMCU 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 SMCU records the requester of the transaction resulting in the error in SELOG[23:16] and interrupts the core.

If the SMCU detects an ECC error during a read or write cycle, SMCISR[0] is set to 1. Whenever the SMCU toggles the SMCISR[0] bit from 0 to 1, an interrupt is generated to the core.

Table 418 shows how the SMCU responds to error conditions.

Table 418. SMCU Error Response

Error Type	SMCU Action
Single-Bit during a read or write	Fix Error (if ECC error correction enabled in the SECCR)
Multi-bit during a read	Target Abort the Internal Bus transaction
Multi-bit during a write	New ECC is generated with bad data and written to SRAM array. Data location is no longer valid.

Note: If ECC reporting is enabled with SECCR[1] or SECCR[0] and an ECC error occurs, SMCISR[0] is set and SELOG/SECAR/SECUAR logs the error in addition to Table 418 actions.



8.4.1 Single-Bit Error Detection

When enabled, the SMCU interrupts the core when the ECC logic detects a single-bit error by setting the appropriate bit in the MCISR register. The core knows the interrupt was caused by a single-bit error by polling the SELOG register. The SRAM Control Block ensures that correct data is returned but the interrupt handler is responsible for scrubbing the error in the array (refer to [Section 8.3.3.4, “Scrubbing” on page 690](#)).

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 SMCU.
- SMCU fixes the error prior to returning the data.
- SMCU clears SELOG[8] indicating a single-bit error.
- SMCU records requester of transaction that resulted in an error in SELOG[23:16]
- The SMCU loads SELOG[7:0] with the syndrome that indicated the error.
- The SMCU loads SECAR[31:2] and SECUAR with address where the error occurred.
- Since the core needs to scrub the error in the array, the SMCU sets MCISR[0] to 1 (assuming it is not already set).
 - Setting any bit in the MCISR causes an interrupt to the core.
- Software polls interrupt status register. Bit 0 = 1 indicates first error has occurred.
- Software polls SELOG, SECAR and SECUAR and scrubs the error at the location specified by SECAR and SECUAR.
- Software writes a 1 to SMCISR[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.

SECAR, SECUAR and SELOG remain registered until software explicitly clears them.

If a second error occurs before software clears the first by resetting SMCISR[0], the error is not logged but the SMCU carries out the action described in [Table 418](#).



8.4.2 Multi-bit Error Detection

If a multi-bit error occurs during a read or write transaction and error reporting is enabled, the SMCU sets SMCISR[0] 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 SELOG registers.

When SMCU detects a multi-bit error during a read cycle and ECC calculation is enabled in the SECCR, the SMCU target aborts the transaction, indicating to the MCU port that an unrecoverable error has been detected. When the SMCU port is the north internal bus port, the north internal bus port notifies the internal bus initiator of a multi-bit error by returning a target abort. The SMCU records the error type in SELOG and the address in SECAR and SECUAR.

When SMCU detects a multi-bit error during a write²⁵ cycle and error reporting is enabled in the SECCR, the SMCU records the first multi-bit error by programming SELOG, SECAR and SECUAR. The SMCU generates new ECC with the data before sending it on **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 SMCISR[0], the error is not logged but the SMCU carries out the action described in [Table 418](#).

It is the interrupt handler responsibility to decide how to handle this error condition and clear the SMCISR.

25. Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See [Section 8.3.3.1, "ECC Generation" on page 684](#) for details.



8.5 Parity Interrupts/Error Conditions

If a data parity error is detected on any of the SMCU ports and parity is enabled, the SMCU records the requesting port that detected the parity error in the SPCSR[19:16] and interrupts the core. Refer to the [Section 8.6.8, "SRAM Parity Control and Status Register — SPARCSR" on page 706](#)

When the SMCU detects a parity error, the SMCISR[8] is set to 1. Whenever the SMCU toggles the SMCUSR[8] bit from 0 to 1, an interrupt is generated to the core.



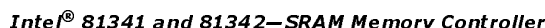
8.6 Register Definitions

A series of configuration registers control the SMCU. Software can determine the status of the SMCU by reading the status registers. [Table 419](#) lists all of the SMCU registers which are detailed further in proceeding sections.

Note: Constant polling of SMCU MMRs can result in inducing long latencies in peripheral unit SRAM transactions, and therefore may negatively impact performance. Polling of SMCU MMRs should be avoided.

Table 419. Memory Controller Register

Section, Register Name — Acronym (Page)
Section 8.6.1, “SRAM Base Address Register — SRAMBAR” on page 700
Section 8.6.2, “SRAM Upper Base Address Register — SRAMUBAR” on page 700
Section 8.6.3, “SRAM ECC Control Register — SECR” on page 701
Section 8.6.4, “SRAM ECC Log Register — SELOGR” on page 702
Section 8.6.5, “SRAM ECC Address Register — SEAR” on page 704
Section 8.6.6, “SRAM ECC Context Address Register — SECAR” on page 704
Section 8.6.7, “SRAM ECC Test Register — SECTST” on page 705
Section 8.6.8, “SRAM Parity Control and Status Register — SPARCSR” on page 706
Section 8.6.9, “SRAM Parity Address Register — SPAR” on page 707
Section 8.6.10, “SRAM Parity Upper Address Register — SPUAR” on page 707
Section 8.6.6, “SRAM ECC Context Address Register — SECAR” on page 704
Section 8.6.11, “SRAM Memory Controller Interrupt Status Register — SMCISR” on page 708



This register indicates the lower twelve bits of the beginning address (base address) of SRAM memory array space. The SRAM is addressed using a 36-bit address. This register is used in conjunction with the [Section 8.6.2, SRAM Upper Base Address Register — SRAMUBAR](#). After reset the default starting address of the SRAM memory is 0_FFE0_0000H.



8.6.3 SRAM ECC Control Register — SECR

This register programs the SMCU 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

For more details, see [Section 8.3.3, "Error Correction and Detection"](#) on page 683 and [Section 8.4, "ECC Interrupts/Error Conditions"](#) on page 695.

Table 422. SRAM ECC Control Register — SECR

Intel XScale® processor Local Bus Address Offset +1508H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:04	000 0000H	Reserved
03	1 ₂	Read-only as 1 ₂ .
02	0 ₂	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
01	0 ₂	Multi-Bit Error Reporting Enable: Enables or disables the reporting (interrupt generation) of a multi-bit error condition. 0 = Disable multi-bit error reporting 1 = Enable multi-bit error reporting
00	0 ₂	Single Bit Error Reporting Enable: Enables or disables the reporting (interrupt generation) of a single bit error condition. 0 = Disable single bit error reporting 1 = Enable single bit error reporting



8.6.4 SRAM ECC Log Register — SELOGR

The SRAM ECC Log Register is responsible for logging the error types detected on the local memory bus. One error 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 not correctable (see Table 416, “Syndrome Decoding” on page 686).

The error recorded in SELOGR corresponds to the address in SECAR and SECUAR.

The SELOGR comprise read-only bits and only have meaning if SMCISR[0] or SMCISR[1] is non-zero. For more details on error handling, see Section 8.3.3, “Error Correction and Detection” on page 683.

Table 423. SRAM ECC Log Register — SELOG (Sheet 1 of 2)

31 28 24 20 16 12 8 4 0 IOP Attributes ro ro ro rv rv rv rv ro ro ro ro ro ro rv rv rv ro rv rv rv ro <		

**Table 423. SRAM ECC Log Register — SELOG (Sheet 2 of 2)**

31 28 24 20 16 12 8 4 0 IOP Attributes ro ro ro rv rv rv rv ro ro ro ro ro ro rv rv rv ro rv rv rv ro ro ro ro ro ro ro PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor Local Bus Address Offset +150CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
19:16	0000 ₂	Direct Memory Port ID: Indicates the direct memory port associated with the ECC logged error. Port ID Port Name 0000₂ North Internal Bus Port 0001₂ Reserved 0010₂ Reserved 0011₂ Reserved 0100₂ Reserved 0101₂ Reserved 0110₂ Reserved 0111₂ Reserved 1000₂ Reserved 1001₂ Reserved 1010₂ Reserved All other IDs are reserved.
15:13	000 ₂	Reserved
12	0 ₂	Read or Write: Indicates if the error occurred during a read or write transaction. 0 = Read error 1 = Write Error
11:09	000 ₂	Reserved
08	0 ₂	ECC Error Type: Indicates the type of error that occurred at this address. 0 = Single Bit Error 1 = Multi-Bit Error
07:00	00H	Syndrome: Holds the syndrome value that indicated the error.



8.6.5 SRAM ECC Address Register — SEAR

This register is responsible for logging the address where the error was detected on the local memory bus. One error can be detected and logged. The software knows which SRAM address had the error by reading this register and decoding the syndrome in the log register. The upper 4 bits are captured in the SECR — refer to [Section 8.6.3, SRAM ECC Control Register — SECR](#). For error details, see [Section 8.3.3, "Error Correction and Detection"](#) on page 683).

Table 424. SRAM ECC Address Register — SEAR

Intel XScale® processor Local Bus Address Offset +1510H Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:02	0	Error Address: Stores the lower 30 bits of the address that resulted in a single bit or multi-bit error.
01:00	00 ₂	Reserved

8.6.6 SRAM ECC Context Address Register — SECAR

This register is responsible for logging the descriptor tag of the descriptor while the ECC error was detected on the local memory bus. One error can be detected and logged. The software knows which descriptor was being processed by reading this register.

Table 425. SRAM ECC Context Address Register — SECAR

Intel XScale® processor Local Bus Address Offset +1514H Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:24	00H	Reserved
23:00	00 0000H	Reserved.

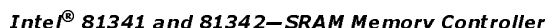


8.6.7 SRAM ECC Test Register — SECTST

This register allows testing between the SRAM ECC logic and the memory subsystem (Section 8.3.3.6, “ECC Testing” on page 691). 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.

Table 426. SRAM ECC Test Register — SECTST

Bit	Default	Description
31:07	00 0000H	Reserved
06:00	00H	ECC Mask: 7-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 Section 8.3.3.6, “ECC Testing” on page 691.



This register programs the SMCU parity checking capabilities. This register is also responsible for logging the error types detected on the SMCU memory ports. Only one error can be detected and logged. The error recorded corresponds to the addresses in (SPAR, SPUAR) and (SPCAR, SPCUAR).



8.6.9 SRAM Parity Address Register — SPAR

This register is responsible for logging the lower 32-bit address of where the error was detected on the SMCU memory ports. Note that the address is 36-bit. This register is used in conjunction with the [Section 8.6.10, “SRAM Parity Upper Address Register — SPUAR” on page 707](#). One error can be detected and logged. The software knows which SRAM address had the error by reading this register and decoding contents of associated log register. For error details, see [Section 8.3.3, “Error Correction and Detection” on page 683](#)).

Table 428. SRAM Parity Address Registers — SPAR

Bit	Default	Description																													
31:02	0000 0000H	Error Address: Stores the upper 30 bits of the address that resulted in a parity error.																													
01:00	00 ₂	Reserved																													

8.6.10 SRAM Parity Upper Address Register — SPUAR

This register is responsible for logging the upper 4-bit address of where the error was detected on the SMCU memory ports. Note that the address is 36-bit. This register is used in conjunction with the [Section 8.6.9, “SRAM Parity Address Register — SPAR” on page 707](#). One error can be detected and logged. The software knows which SRAM address had the error by reading this register and decoding contents of associated log register. For error details, see [Section 8.3.4, “Byte Parity Checking and Generation” on page 692](#)).

Table 429. SRAM Parity Upper Address Register — SPUAR

Bit	Default	Description																													
31:04	0000 000H	Reserved																													
03:00	0000 ₂	Parity Error Address: Stores the upper 4 bits of the 36-bit address that resulted in the parity error.																													



8.6.11 SRAM Memory Controller Interrupt Status Register — SMCISR

Setting the SMCISR 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 SMCU has five interrupt conditions: first ECC error (SMCISR[0]), more than one ECC error (SMCISR[1]), first parity error (SMCISR[8]), and more than one parity error (SMCISR[9]).

If the SMCU detects an ECC error and SMCISR[0] is cleared, the error is logged in SELOG and SMCISR[0] is set to 1. If SMCISR[0] is not cleared, any additional ECC errors are not logged and SMCISR[1] is set.

Similarly, if the SMCU detects a parity error and SMCISR[8] is cleared, the parity error is logged in SPCSR and SMCISR[8] is set to 1. If SMCISR[8] is not cleared, any additional parity errors are not logged and SMCISR[9] is set.

Table 430. SRAM Memory Controller Interrupt Status Register — SMCISR

Intel XScale® processor Local Bus Address Offset +152CH Attribute Legend: RW = Read/Write, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
31:05	0000 000H	Reserved
09	0 ₂	Parity N: Indicates that the SMCU detected a Parity error while SMCISR[8] was set. 0 = No error detected 1 = Error detected
08	0 ₂	Parity Error: Indicates that the SMCU detected a Parity error and recorded the error in SPCSR. 0 = No error detected 1 = Error detected and recorded in SPLOG
07:04	0H	Reserved
03	0 ₂	Address Range Error: Indicates that a transaction to an invalid address range. For example, an address that is outside the 1.0 MBytes SRAM space. 0 = No error detected 1 = Error detected
02	0 ₂	Reserved
01	0 ₂	ECC Error N: Indicates that the SMCU detected an ECC error while MCISR[0] was set. 0 = No error detected 1 = Error detected
00	0 ₂	ECC Error O: Indicates that the SMCU detected an ECC error and recorded the error in SELOG. 0 = No error detected 1 = Error detected and recorded in SELOG



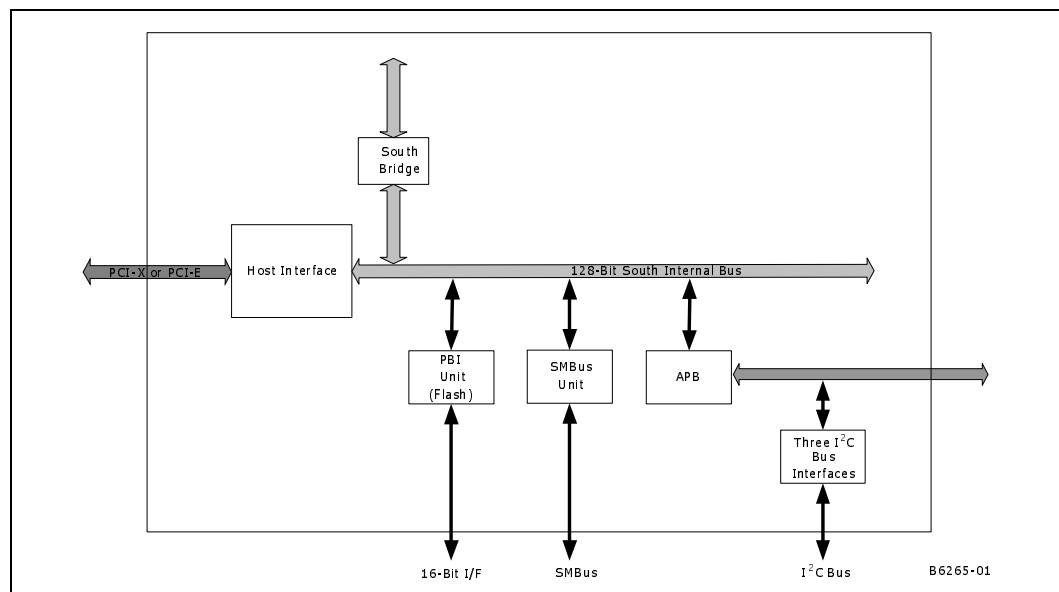
9.0 Peripheral Bus Interface Unit

This chapter describes the Peripheral Bus Interface Unit (PBI) of the Intel® 81341 and 81342 I/O Processors (81341 and 81342). It explains the following:

- Peripheral Bus signals, which consist of address/data, control/status
- Peripheral Bus Read, and write transactions
- Peripheral Bus configuration and Flash Memory Support
- Registers

This chapter also serves as a starting point for the hardware designer when interfacing typical flash components to the 81341 and 81342 Peripheral Bus.

Figure 106. The Peripheral Bus Interface Unit





9.1 Overview

The Peripheral Bus Interface Unit (PBI) is a data communication path to the flash memory components and peripherals of a 81341 and 81342 hardware system. The PBI allows the processor to read and write data to these supported flash components and other peripherals. To perform these tasks at high bandwidth, the PBI bus features a burst read transfer capability which allows successive data transfers for multi-byte read requests. The maximum PBI bus read burst size is four data transfers, regardless of bus width. This means that some read requests made to the PBI may result in multiple PBI bus burst accesses. For example a read request for 32 bytes on a 16-bit PBI bus results in 4 bursts of four 16-bit words on the Peripheral Bus. Multi-byte read requests and multi-byte write requests are supported differently by the PBI. Write requests are limited to a maximum of 4 bytes only and must not span a DWORD boundary. The PBI signals an address error when a write request has its byte-count out of range.

The peripheral bus is controlled by the on-chip bus masters: the Intel XScale® processor 0, the Intel XScale® processor 1, the ATU-E, the ATU-X, and the ADMA units.

The address and data paths are demultiplexed, and the bus width is programmable to 8-, and 16-bit widths. The PBI performs the necessary packing and unpacking of bytes to communicate properly across the 81341 and 81342 Internal Bus.

The PBI unit includes two chip enables. The PBI chip enables activate the appropriate peripheral device when the address falls within one of the PBIs two programmable address ranges. Both address ranges incorporate functionality that optimizes an interface for Flash Memory devices. Each chip enable can support up to 32 MBytes of addressability. For example, there are 25 address signals provided on the PBI interface.



9.2 Peripheral Bus Signals

Bus signals consist of three groups:

- address
- data
- control/status

9.2.1 Address Signal Definitions

The address signal group **A[24:0]** consists of 25 lines which allows the PBI to address up to 32 MBytes per peripheral device. During an address cycle (T_A), the processor drives **A[24:0]** with the starting address of the bus access. During burst read access the wait cycle (T_W) and the T_D cycle, **A[2:0]** address pins provide incrementing byte addresses.

9.2.2 Data Signal Definitions

The data signal group **D[15:0]** consists of 16 lines. The PBI supports either an 8-bit data bus width on **D[7:0]** or 16-bit data bus width on **D[15:0]**. During the address cycle (T_A), bits **D[1:0]** carry the SIZE of the access.

9.2.3 Control/Status Signal Definitions

The control/status signals control peripheral device enables and direction. All output control/status signals are three-state.

A peripheral read may be either non-burst or burst. A non-burst read ends after one data transfer to a single location.

When the data bus is configured for 16 bits, address bits **A[2:1]** are used to burst across up to four short-words. For an 8-bit data bus, address bits **A[1:0]** are used to burst across up to four bytes.

The Output Enable, **POE#**, is used for burst or non-burst read accesses to a peripheral device and is asserted during the $T_A/T_W/T_D$ states.

The Write Enable, **PWE#**, is used for non-burst write accesses to a peripheral device and is asserted during the T_W/T_D states.

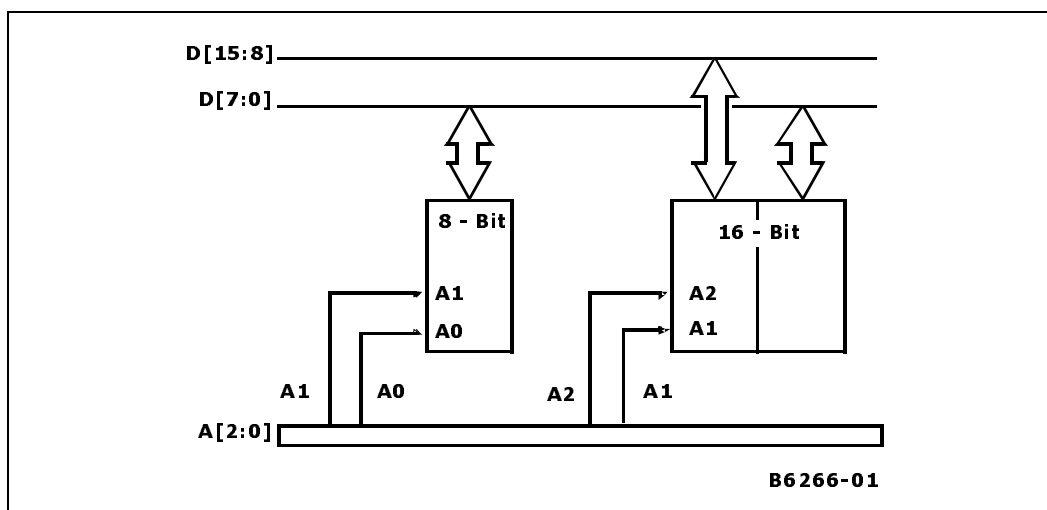
The PBI Reset, **PB_RSTOUT#**, is used to reset the peripheral device. It has the same timings as the internal bus reset signal.

Note: Burst write accesses are not supported by the PBI bus. A multi-byte write request made to the PBI translates into multiple single data write transactions on the PBI bus. For example, each write transaction on the PBI bus ends after one data transfer to a single address location. Note that the number of single data write transactions initiated on the PBI bus are dependent to the PBI bus width.

9.2.4 Bus Width

Each address range's attributes are programmed in the PBI's boundary registers. The PBI allows an 8-, or 16-bit data bus width for each range. The PBI places 8- and 16-bit data on low-order data signals, simplifying the interface to narrow bus external devices. As shown in Figure 107, 8-bit data is placed on lines **D[7:0]**; 16-bit data is placed on lines **D[15:0]**.

Figure 107. Data Width and Low Order Address Lines



The user needs to wire up the flash memories in a manner consistent with the programmed bus width:

- 8-bit region: **A[1:0]** provide the demultiplexed byte address for a read burst.
- 16-bit region: **A[2:1]** provide the demultiplexed short-word address for a read burst.

During initialization, bus width is selected for each of the two address ranges in the Peripheral Base Address Registers (PBBAR0 — PBBAR1). In addition, the PBBAR0-PBBAR1 can be used to configure these ranges as Peripheral Windows and to set a Wait state profile.

The PBI drives determinate values on all address/data signals during T_W/T_D write operation states. For an 8-bit bus, the PBI continues to drive address on unused data signals **D[15:8]**.



9.2.5 Detailed Signal Descriptions

Bus signal descriptions are detailed in [Table 432](#).

Table 431. Bus Signal Descriptions

NAME	DESCRIPTION
D[15:0]	DATA BUS carries 16-bit physical addresses and 8-, or 16-bit data to and from memory. During a data (T_d) cycle, bits 0-7, or 0-15 contain read or write data, depending on the corresponding bus width. During write operations to 8-bit wide memory regions, the PBI drives unused bus pins high or low.
A[24:0]	ADDRESS BUS 24:0 carries the 25-bit address bus which allows the PBI to address up to 32 MBytes per peripheral device. During an address (T_a) cycle, bits A[2:0] contains the starting address of the access. During a burst read data (T_d) cycle, A[2:0] represents the current byte address in the burst transaction. Address bits A[24:3] provide the upper address of the current access and is a constant during the address (T_a), wait state (T_w) and data cycles (T_d) cycles. A[2:1] are used for an 16-bit wide peripheral while A1:0 are used for an 8-bit wide peripheral.
POE#	PERIPHERAL OUTPUT ENABLE specifies, during a T_a cycle, whether the operation is a write (1) or read (0). It is latched on-chip and remains valid during T_d cycles. This signal is used as an OUTPUT ENABLE signal (OE#) for Peripheral Devices.
PCE[1:0]#	PERIPHERAL CHIP ENABLES 1:0 specify, during a T_a cycle, which of the two Memory Address Ranges are associated with the current bus access. It remains valid during T_d cycles
PWE#	PERIPHERAL WRITE ENABLE indicates to a peripheral device whether or not to use the data on the D15:0 bus to write the addressed space. It is low during T_w cycles and deasserts during the T_d cycle for a write; it is high during T_a and T_w/T_d cycles for a read.
PB_RSTOUT#	PERIPHERAL BUS RESET can be used to reset the peripheral device. PB_RSTOUT# has the same timings as the internal bus reset signal.

9.2.6 Flash Memory Support

PBI peripheral bus interface supports 8-, or 16- bit Flash devices.

The PBI provides programmable wait state functionality for peripheral memory windows.

Note: Potentially, programmable wait state functionality could be connected to any peripheral device that has a deterministic wait state profile. However, data valid and turn-around times would need to fit within parameters provided by programmable wait state profiles to support Flash devices.

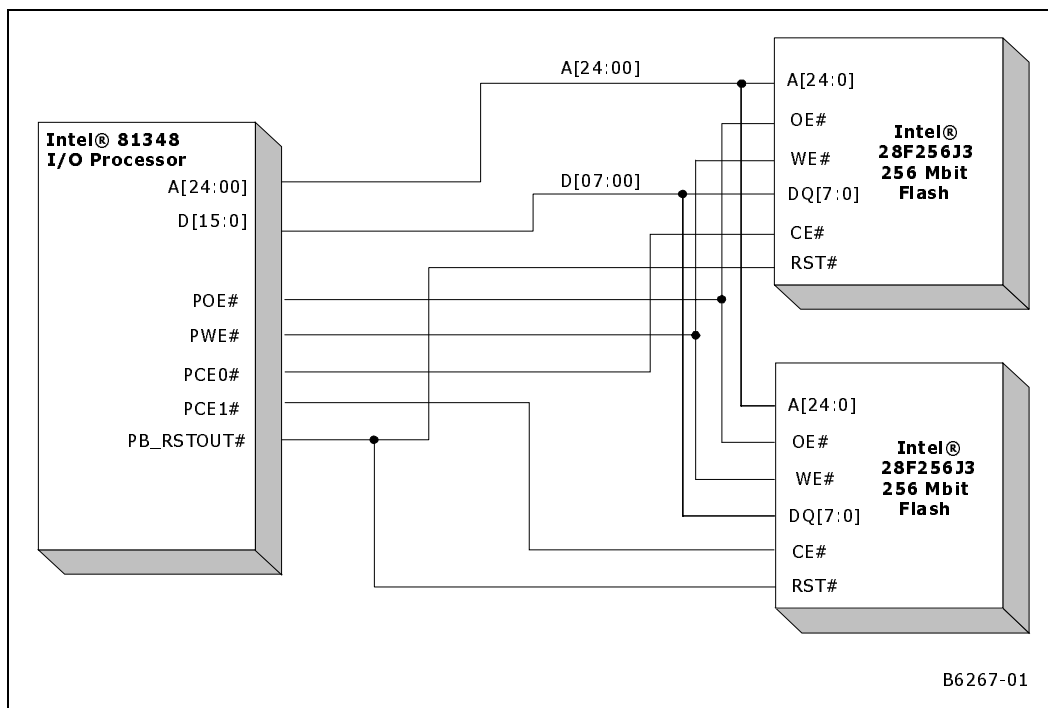
Any write transactions issued to a Flash address space window must always represent a single flash bus data cycle (**strb**, **strh**).

The peripheral chip enables, **PCE[1:0]#**, activate the appropriate Peripheral window when the address falls within one of the Peripheral address ranges.

Note: By default, bank 0 is enabled with the maximum number of Address-to-Data and Recovery Wait states. The width of the interface can be strapped for either 8-bit wide Flash or 16-bit wide flash. Thus, **PCE0#** is the Peripheral Bus chip enable to be used for booting purposes.

Figure 108 on page 714 illustrates how two 8-bit Flash devices would interface with an Intel XScale® processor through the PBI Interface.

Figure 108. Sixty-Four Mbyte Flash Memory System





9.2.6.1 Flash Read Cycle

Reading a Flash device involves driving the address, output enable, and chip enable. Depending on the speed of the Flash device, the data returns several cycles later.

The definition of address-to-data wait states are the number of cycles between the assertion of **PCE[1:0]#**, and the arrival of data from the Flash device on **D[7:0]** (8-bit Flash). The definition of recovery wait states are the number of cycles between the data arrival on **D[7:0]** and the address for the next Peripheral transaction.

Address-to-data and recovery wait states are programmed in PBBAR0 and PBBAR1 and are identical for reads and writes. Since the read wait state requirement is typically greater, the write wait state requirement is insured to be met.

Figure 109 illustrates a single read cycle example for a 120 ns Flash device. The number of wait states used for address-to-data is provided by the *Address-to-Data Wait States* field in the PBBARx.

Figure 109. 120 ns Flash Single Transfer Read Cycle

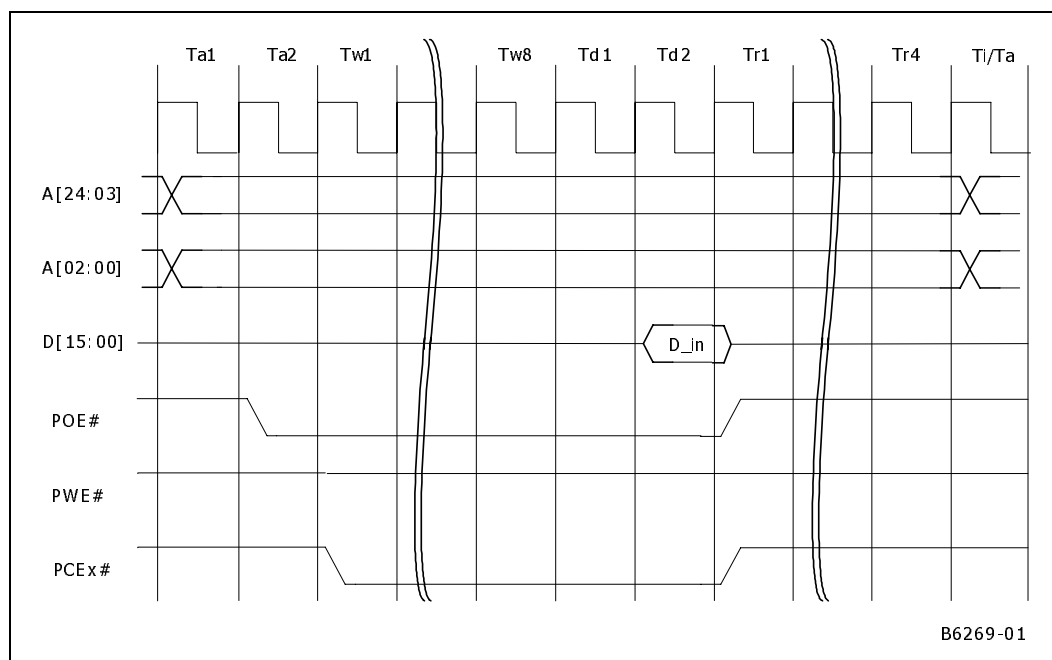
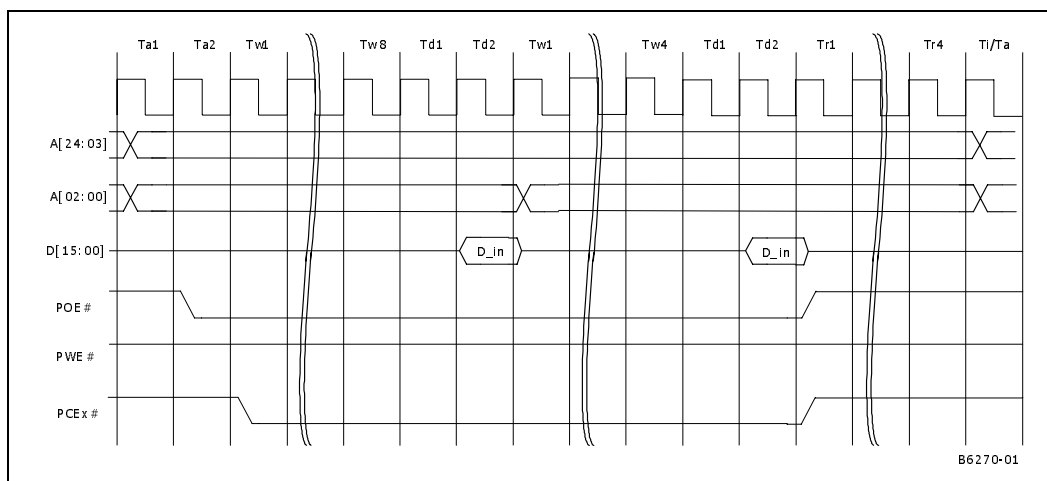




Figure 110 illustrates a burst read cycle example for a 120 ns Flash device. This example is illustrating a burst of two bytes or words. The PBI is capable of bursting up to four bytes or words. The number of wait states used for address-to-data and data-to-data are provided by the *Address-to-Data Wait States* and the *Data-to-Data Wait States* field in PBBARx respectively.

Figure 110. 120 ns Flash Burst Read Cycle



Refer to Table 432 for the programmable address-to-data, data-to-data, and recovery wait states. These numbers are based on a 66 MHz internal clock for the PBI interface.

Table 432. Flash Wait State Profile Programming¹

Flash Speed	Address-to-Data Wait States	Data-to-Data Wait States ²	Recovery Wait States
<= 55 ns	4	4	1
<= 120 ns	8	4	4
<= 150 ns	12	4	4

Notes:

- Each Wait State represents a 15 ns period based on a 66 MHz clock. Refer to the appropriate Flash device datasheets for programming accurate wait state numbers.
- Data-to-Data wait states are used for burst reads.



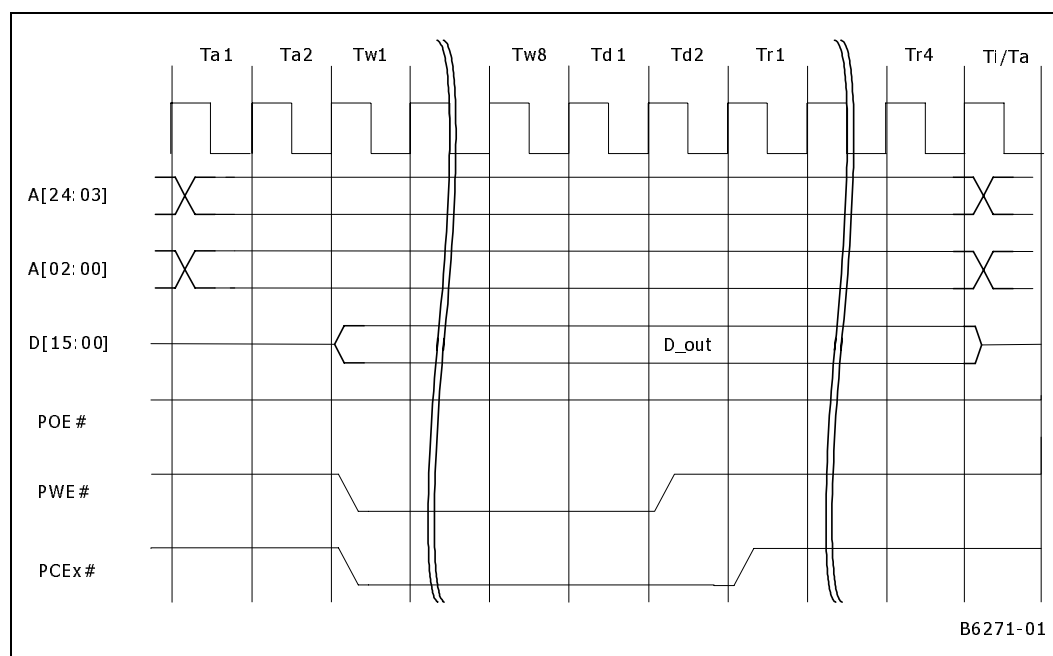
9.2.6.2 Flash Write Cycle

Address-to-data and recovery wait states for reads and writes are identical and programmed in PBBAR0 and PBBAR1. Refer to [Table 432](#) for the programmable address-to data wait states. However, Any write transactions issued to a Peripheral address space window must always represent a single peripheral bus data cycle (**strb**, **strh**) depending on the bus width selected in PBBAR0 — PBBAR1.

The PBI supports multi-byte write requests from the internal bus agents. Multi-byte read and write requests are supported differently by the PBI. Write requests are limited to a maximum of four bytes only and must not span a DWORD boundary. For a write request with an address and byte-count combination that spans a DWORD boundary, the PBI signals an address error and set bit 0 of the “PBI Status Register — PBISR” on [page 719](#). Unlike multi-byte read requests, the PBI supports multi-byte write requests by breaking the writes on the PBI bus into multiple single data write transactions. The number of single data write transactions initiated on the PBI bus are dependent to the PBI bus width. For example, an aligned DWORD write on an 8-bit PBI bus turns into four single 8-bit write transactions on the PBI bus, and an aligned DWORD write request on a 16-bit PBI bus turns into two single 16-bit write transactions on the PBI bus. [Figure 111](#) shows the only type of write transaction supported by the PBI bus. For example, each write transaction ends after one data transfer to a single address location regardless of bus width.

[Figure 111](#) illustrates a write cycle example for a 120 ns Flash device. Both 8- and 16-bit wide bus timings are identical.

Figure 111. 120 ns Flash Single Write Cycle¹



Notes:

1. The PBI bus does not burst write transactions. A multi-byte write request made to the PBI is translated into multiple single data write transactions on the PBI bus. And the number of single data write transactions initiated on the PBI bus are dependent to the PBI bus width.



9.3 Register Definitions

A series of configuration registers control PBI. Software can determine PBI status by reading the status register. [Table 433](#) lists all PBI registers which are detailed further in proceeding sections.

Table 433. Peripheral Bus Interface Registers

Section, Register Name — Acronym (Page)
Section 9.3.1, "PBI Control Register — PBCR" on page 719
Section 9.3.2, "PBI Status Register — PBISR" on page 719
Section 9.3.4, "PBI Base Address Register 0 — PBBAR0" on page 721
Section 9.3.5, "PBI Limit Register 0 — PBLR0" on page 722
Section 9.3.6, "PBI Base Address Register 1 — PBBAR1" on page 723
Section 9.3.7, "PBI Limit Register 1 — PBLR1" on page 724
Section 9.3.8, "PBI Drive Strength Control Register — PBDSCR" on page 725
Section 9.3.9, "Processor Frequency Register - PFR" on page 726
Reserved.
Reserved.
Section 9.3.10, "External Strap Status Register 0 — ESSTR0" on page 727
Reserved.
Reserved.
Section 9.3.12, "Unique ID Register 1 — UID1" on page 728



9.3.1 PBI Control Register — PBCR

The PBI Control Register (PBCR) is responsible for enabling operation of PBI state machines.

Table 434. PBI Control Register — PBCR

		31 28 24 20 16 12 8 4 0 IOP Attributes rv rv rv rv rw rw rw rw rv rv rv rv rw rw rw rw rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv rv na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na PCI Attributes Intel XScale[®] processor Local Bus Address Offset +1580H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																															
31:28	0H	Reserved.																															
27:24	0H	PBI Memory Window 1 Upper 4-bit address. These bits, with PBBAR1 form a 36-bit base address.																															
23:20	0H	Reserved.																															
19:16	0H	PBI Memory Window 0 Upper 4-bit address. These bits, with PBBAR0 form a 36-bit base address.																															
15:04	000H	Reserved.																															
03	0 ₂	Reserved.																															
02:01	00	Reserved (bit[2] reads as 1).																															
00	1 ₂	PBI Enable: When set, this bit enables the PBI unit bus interface 0 = Peripheral Bus Disabled 1 = Peripheral Bus Enabled.																															



9.3.3 Determining Block Sizes for Memory Windows

The memory window size can be determined by writing ones to the appropriate upper bits of the limit register. The binary-weighted value of the first non-zero bit set in the limit register indicates the size of the memory window. Table 436 describes the relationship between limit register values and the byte sizes of the memory window.

Table 436. Memory Block Size Limit Register Values

Limit Register Value ^a	Size (in Bytes)	Limit Register Value	Size (in Bytes)
FFFFF000H	4K	FF800000H	8 M
FFFFE000H	8K	FF000000H	16 M
FFFFC000H	16K	FE000000H	32 M
FFFF8000H	32K	FC000000H	Address Window Closed.
FFFF0000H	64K	F8000000H	
FFFE0000H	128K	F0000000H	
FFFC0000H	256K	E0000000H	
FFF80000H	512K	C0000000H	
FFF00000H	1 M	80000000H	
FFE00000H	2 M	00000000H	
FFC00000H	4 M		

a. Smallest Limit Register Value is 4 KBytes.

As an example, assume that FFF0 0000H is written to the PBI Limit Register 0 (PBLR0). Scanning upwards starting at bit 12, bit 20 is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicating a 1 Mbyte of memory window.

When programming the Base and Limit Registers for a memory window, the Base Address always needs to be aligned the size of the memory window set in a limit register. For a 1 Mbyte memory window, only bit 20 through bit 31 of the base address from the PBI Base Address Register 0 (PBBAR0) are relevant to the PBI when decoding Memory Window 0.

Warning: A given PBI Base (PBBAR0-PBBAR1) and Limit (PBLR0-PBLR1) register pair should not be modified during the time there is activity on the peripheral bus associated with that particular peripheral memory window. For instance, following boot-up, code executing from Peripheral Memory Window 0 may be used to modify the PBI Base and Limit registers for Peripheral Memory Window 1, but **not** for Peripheral Memory Window 0.



9.3.4 PBI Base Address Register 0 — PBBAR0

The PBI Base Address Register 0 (PBBAR0) defines the block of memory addresses where PBI Memory Window 0 begins. The PBBAR0 defines the base address and describes the required memory block size; see [Section 9.3.3, “Determining Block Sizes for Memory Windows” on page 720](#). The selected base address needs to be naturally aligned to the granularity of the memory block size. For instance, when a 64 Kbyte memory window size is selected, the base address needs to be 64 Kbyte address aligned (i.e., bits 15:12 of the base address are required to be 000b).

Bits 1:0 define the PBI bus width for PBI Memory Window 0.

Table 437. PBI Base Address Register 0 — PBBAR0



9.3.5 PBI Limit Register 0 — PBLR0

The 81341 and 81342 limit register's (PBLR0) programmed value must be naturally aligned with the base address register's (PBBAR0) programmed value. The limit register is used as a mask when the address decode for memory window 0 is performed.

Table 438. PBI Limit Register 0 — PBLR0

Intel XScale® processor Local Bus Address Offset + 158CH Attribute Legend: RW = Read/Write, RV = Reserved, PR = Preserved, RS = Read/Set, RC = Read Clear, RO = Read Only, NA = Not Accessible		
Bit	Default	Description
31:12	FE00H	Memory Window 0 Limit: This value determines the memory block size required for the Memory Window 0. Defaults to a 32MB Peripheral Window
11:00	000H	Reserved



9.3.6 PBI Base Address Register 1 — PBBAR1

The PBI Base Address Register 1 (PBBAR1) defines the block of memory addresses where PBI Memory Window 1 begins. The PBBAR1 defines the base address and describes the required memory block size; see [Section 9.3.3, “Determining Block Sizes for Memory Windows” on page 720](#). The selected base address needs to be naturally aligned to the granularity of the memory block size. For instance, when a 64 Kbyte memory window size is selected, the base address needs to be 64 Kbyte address aligned (i.e., bits 15:12 of the base address are required to be 000b).

Bits 1:0 define the PBI bus width for PBI Memory Window 1.

Table 439. PBI Base Address Register 1 — PBBAR1

IOP Attributes 31 28 24 20 16 12 8 4 0 rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rv rw rw rw rw rw PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor Local Bus Address Offset +1590H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:12	00000H	Memory Window 1 Base Address: These bits define the actual location the PBI responds to for accesses to Memory Window 1.
11:09	001 ₂	Data-to-Data Wait States: Defines the number of data-to-data wait states for the Peripheral window during a burst read transaction. 00X - Wait states are equal to the programmed value in the Address-to-Data Wait States field. 010 - 4 Data-to-Data wait states 011 - 8 Data-to-Data wait states 100 - 12 Data-to-Data wait states 101 - 16 Data-to-Data wait states Others - 20 Data-to-Data wait states Note: By default, data-to-data wait states are 20, since it is the same as address-to-data wait states
08:06	111 ₂	Recovery Cycle Wait States: Defines the number of recovery cycle wait states for the Peripheral window. 000 - 1 Recovery wait state 001 - 4 Recovery wait states 010 - 8 Recovery wait states 011 - 12 Recovery wait states 100 - 16 Recovery wait states Others (Default) - 20 Recovery wait states
05	0 ₂	Reserved
04:02	111 ₂	Address-to-Data Wait States: Defines the number of address-to-data wait states for the Peripheral window during a read or write transaction. 000 - 4 Address-to-Data wait states 001 - 8 Address-to-Data wait states 010 - 12 Address-to-Data wait states 011 - 16 Address-to-Data wait states Others (Default) - 20 Address-to-Data wait states
01:00	00 ₂	Bus Width: Defines the bus width for this Memory Window: 00 - 8 bits wide 01 - 16 bits wide 10 - Reserved 11 - Reserved



9.3.7 PBI Limit Register 1 — PBLR1

The 81341 and 81342 limit register (PBLR1) and base address register (PBBAR1) programmed values must be naturally aligned. The limit register is used as a mask when the address decode for memory window 1 is performed.

Table 440. PBI Limit Register 1 — PBLR1

Bit	Default	Description
31:12	00000H	Memory Window 1 Limit: Determines the memory block size required for the Memory Window 1.
11:00	000H	Reserved



9.3.8 PBI Drive Strength Control Register — PBDSCR

This register is used to manually control slew rate and drive strength for all of 81341 and 81342 pins with the exception of high-speed serial interfaces, the SDRAM interface, and the PCI-X interface.

Note: By default, this register is **not** required to program. This register should not be programmed to a different value without consulting the 81341 and 81342 where the appropriate values are specified.

Table 441. PBI Drive Strength Control Register – PBDSCR

Intel XScale® processor Local Bus Address offset
+2080H

Attribute Legend:
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible



9.3.9 Processor Frequency Register - PFR

This register indicates the clock frequencies of the Intel XScale® processor and the internal buses of the 81341 and 81342. For products with two enabled Intel XScale® processors, the cores run at the same clock frequency.

Table 442. Processor Frequency Register - PFR



9.3.10 External Strap Status Register 0 — ESSTS0

The External Strap Status Register 0 provides software a way to read the states of the current settings of the straps. Refer to the Clock and Reset Chapter for external strap descriptions.

Table 443. External Strap Status Register 0 — ESSTS0

Intel XScale® processor Local Bus Address offset +2188H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default ^a	Description	
31:17	0000H	Reserved.	
16	x2	CLK_SRC_PCIE#	
15	x2	INTERFACE_SEL_PCIX#	
14	x2	Reserved.	
13	x2	LK_DN_RST_BYPASS#	
12	x2	PCIX_PULLUP#	
11:10	xx2	Reserved.	
09:07	xxx2	Reserved.	
06	x2	BOOT_WIDTH_8#. Note: This bit reflects the inverted value of the BOOT_WIDTH_8# strap.	
05:04	xx2	MEM_FREQ[1:0]	
03:00	xxxx2	SMBus Address Straps: Register Bit SMBus Address Bit 0 SMB_A1 1 SMB_A2 2 SMB_A3 3 SMB_A5	

a. Default values are product and feature dependent. See bit descriptions for default values.



9.3.11 Unique ID Register 0 — UID0

The Unique ID register 0 represents a 28-bit unique value.

Table 444. Unique ID Register 0 — UID0

			Intel XScale® processor Local Bus Address offset +2194H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description				
31:28	0H	Reserved.				
27:00	xxxx.xxxx.xxxx.xx xx.xxxx.xxxx.xxxx 2	Unique ID 0.				

9.3.12 Unique ID Register 1 — UID1

The Unique ID register 1 represents a 23-bit unique value.

Table 445. Unique ID Register 1 — UID1

			Intel XScale® processor Local Bus Address offset +2198H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description				
31:23	00H	Reserved.				
22:00	xxx.xxxx.xxxx.xxx x.xxxx.xxxx2	Unique ID 1.				



10.0 Exception Initiator and Boot Sequence

This chapter describes the 81341 and 81342 Exception Initiator. It also describes the Intel XScale® processor booting sequence. The operation modes, setup, and implementation of the Exception Initiator are described in this chapter.

10.1 Overview

The 81341 and 81342 supports two Intel XScale® processors, and provides the following features to facilitate the co-existence of two cores:

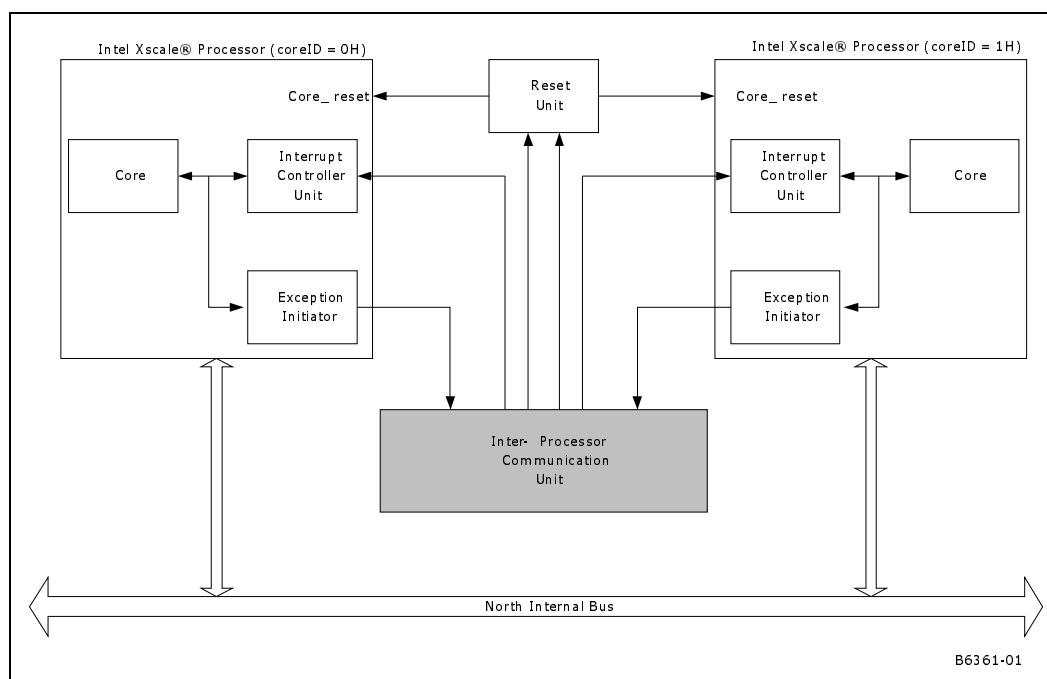
- Core ID
- Inter-Processor Reset
- Inter-Processor Interrupt

Each Intel XScale® processor has a co-processor register in CP6 that contains a unique core identification number (coreID). The coreID is used by software to be able to identify and direct each core's execution path. The coreID is also used for inter-processor reset or interrupt generation. For example, each core can initiate a core reset to another core in the system, including itself. And each core can also generate software interrupts to another core, including itself.

10.2 Theory of Operation

Figure 112 provides a block diagram showing how inter-processor communication is achieved. Inter-processor communication is done using a private interface controlled by the Inter-processor Communication Unit (IPC). There are two types of events that can be issued from one core to another: an interrupt and a reset. The cores can initiate and event to another core by writing its co-processor registers. The IPC monitors issued events from all cores, and then routes the interrupt or reset to the appropriate core. An event is initiated by providing a core identification number (coreID) of the targeted core. The IPC then uses the coreID to know how to route the interrupt or reset.

Figure 112. Inter-Processor Communication Block Diagram





10.2.1 Core Identity

Each core supports a 4-bit field in a co-processor register (CIDR) that contains a unique core identification number (coreID). Refer to [Section 447, “Core Identification Register — CIDR” on page 735](#). On the 81341 and 81342 there are two cores with assigned coreIDs of 0H and 1H. On 81348ArnoldVal Vista the Application core has a coreID equal to 1H, and the Transport core has a coreID equal to 0H. The coreID is also used by hardware to indicate a targeted core for an event.

10.2.2 Targeted Reset

Each core can initiate a reset to the another core, including itself by writing a co-processor register, Targeted Reset Register (TARRSTR) located in CP6. Refer to [Section 450, “Targeted Reset Register — TARRSTR” on page 738](#). TARRSTR register provides a 4-bit field for the coreID that is the target of the reset. By simply writing the TARRSTR register with the appropriate coreID will initiate a reset to the targeted core. The reset logic will assert and de-assert the reset signal to the targeted core for the required duration. For example, software does not have to take any further action after writing the TARRSTR register. In addition to the TARRSTR register, each core also provides another co-processor register, Reset Cause Status Register (RCSR). Refer to [Section 448, “Reset Cause Status Register — RCSR” on page 736](#). This register provides a mechanism for software to identify the cause of the a reset — for example, a system reset or a core initiated reset. Bit 0 of the RCSR register is cleared when the core is reset due to a system reset, and is set when the core is reset due to a targeted reset.

10.2.3 Inter-Processor Interrupt Generation

Each core can initiate software interrupts to another core, including directing the interrupts to itself by writing a co-processor register, Core Interrupt Generation Register (CINTGENR). Refer to [Section 449, “Software Interrupt Generation Register — SINTGENR” on page 737](#). CINTGENR register provides a 4-bit field and an 5-bit field, which contain the targeted coreID number and an interrupt source number (ISN) respectively. On 81341 and 81342 only five bits are implemented for the interrupt source number, thus allowing up to thirty-two unique interrupt source numbers that can be generated by a given core. By simply writing the CINTGENR register with the appropriate coreID and ISN will generate an interrupt to the targeted core. In addition to the CINTGENR register, each core also provides another co-processor register (IIPNDR) in the Interrupt Controller Unit where Core generated interrupts are posted. For example, the 5-bit ISN is decoded and the corresponding bit in the IIPNDR register is set. Bit 0 is set when ISN is equal to 00H, bit 1 is set when ISN is equal to 01H, and so on. Software must write 1's to clear pending interrupt bits in the IIPNDR.



10.3 Intel XScale® Processor Boot Sequence

This section describes various Intel XScale® processors booting sequence options, and how these various booting options are accomplished. The default boot device is the Flash memory, but the 81341 and 81342 also provides a mechanism which can allow each core to boot from other memory subsystem (in other words, SRAM memory.)

Caution: SRAM memory must be properly scrubbed before being used.

10.3.1 Theory of Operation

The 81341 and 81342 provides hardware resources that can be used to accomplish various boot sequence options.

- **HOLD_X0_IN_RST#** — this is an external pin strap that can be used to control whether coreID0 stays in reset after the system reset is de-asserted. When the strap is pulled low, coreID0 stays in reset after the main system reset is de-asserted. The status of **HOLD_X0_IN_RST#** is reflected in bit 0 of the ATU PCI Configuration and Status Register (PCSR). Software can only clear this bit and is required to clear this bit to de-assert coreID0 reset.
- **HOLD_X1_IN_RST#** — this is an external pin strap that can be used control whether coreID1 stays in reset after the system reset is de-asserted. When the strap is pulled low, coreID1 stays in reset after the main system reset is de-asserted. The status of **HOLD_X1_IN_RST#** is reflected in bit 1 of the PCI Configuration and Status Register (PCSR). Software can only clear this bit and is required to clear this bit to de-assert coreID1 reset.
- **Software Interrupt Pending Register** — Initialization software running on slave cores can synchronize with a master core by using their corresponding software interrupt pending registers. Each core/ICU consists of such a register. Note that although the software interrupt pending register is not designed for that purpose, it provides the correct attributes for that purpose. The master core would be the core that starts running and carries out adequate system initialization before other slave cores are allowed to proceed with their initialization sequence. The system may want one core (a master core) to initialize part of the system while the other cores (slave cores) would wait in a software loop — wait on the software interrupt pending register. When the master core has completed adequate initialization, it can release the slave cores by writing the software interrupt pending registers of the slave cores.

The next sections describe the steps involved in the various boot sequence for 81341 and 81342.



10.3.2 81341 and 81342 Boot Sequence

The next sections describe the boot sequence options for 81341 and 81342. 81341 and 81342 consists of two Intel XScale® processors (coreID0 and coreID1).

10.3.2.1 Boot Sequence — coreID0 and coreID1 simultaneous start running from Flash Memory

The following steps can apply to either core running first. For example, there is no restriction on which core starts running first. Software can control how to synchronize the two cores. This is a scenario where both cores are allowed to run after reset is deasserted. This example also demonstrates how one core (coreID0) in the system can act as a master core and be able to synchronize with the slave core (coreID1) during the initialization sequence. The synchronization is done by software using the coreIDs.

1. Pull external straps HOLD_X0_IN_RST# and HOLD_X1_IN_RST# high in order to allow coreID0 and coreID1 to start running after reset is deasserted.
2. coreID0 (master core) would run by initializing adequate elements of the system, while coreID1 (slave core) would also run but eventually wait in a software loop — waiting for the software interrupt pending register to be written by the master core.
3. After the master core (coreID0) has completed enough initialization, it can initiate a software interrupt to the slave core (coreID1) which will cause the software interrupt pending register to alter. The slave core would then exit the software loop and proceed with its normal execution based on the content of the software interrupt pending register.

10.3.2.2 Boot Sequence — Both coreID0 and coreID1 run from SRAM Memory (no Flash Memory)

The following steps can apply to either core running first. For example, there is no restriction on which core starts running first. Software can control how to synchronize the two cores.

1. Pull external straps HOLD_X0_IN_RST# and HOLD_X1_IN_RST# low in order to hold coreID0 and CoreID1 in reset.
2. The Host processor, via the host interface, would scrub the SRAM memory, download code into SRAM, and setup the SRAM Base Address Registers to 0 0000 0000H. This would allow the core access to be claimed by the SRAM MCU during boot.
3. The Host processor would then clear the HOLD_X0_IN_RST# bit while still keeping coreID1 in reset.
4. coreID0 would start to run from SRAM memory, initialize the system.
5. coreID0 would then clear the HOLD_X1_IN_RST# bit to release coreID1 from reset.
6. coreID1 would start running from SRAM Memory address.



10.3.2.3 Boot Sequence — coreID0 runs from Flash Memory and coreID1 runs from SRAM Memory

1. Pull the external strap HOLD_X1_IN_RST# low to hold coreID1 in reset, while pulling the external strap HOLD_X0_IN_RST# high to allow coreID0 to run.
2. coreID0 will start running after reset from Flash Memory.
3. coreID0 would scrub the SRAM memory, and download code image for coreID1 into SRAM memory.
4. coreID0 will setup the SRAM Base Address Registers to 0 0000 0000H. This would allow the core access to be claimed by the SRAM MCU during boot.
5. coreID0 would then clear the HOLD_X1_IN_RST# bit to release coreID1 from reset.
6. coreID1 would start running from SRAM Memory address.

Note: The north interface of the internal bus bridge performs subtractive decoding. Normally Flash (address starting at 0 0000 0000H) access by the core would be claimed by the bridge north interface and forwarded to the south internal, where the Flash device is located. The SRAM Memory Controller resides on the North internal bus. If the SRAM Base Address Registers are setup to 0 0000 0000H by the Host processor via the Host Interface, the core can then boot from the SRAM Memory.



10.4 Register Definitions

All the Exception Initiator registers are 32-bit and visible as Intel XScale® processor co-processor CP6 registers. The coprocessor registers may be accessed/manipulated with the MCR, MRC, STC and LDC instructions. There are 256 registers per coprocessor, organized in 16 groups of 16 registers. The CR_m field of the instruction denotes the register group to access. And the CR_n field of the instruction denotes the register number within that group.

Table 446. Exception Initiator Register Addresses

Register Name	Description	Coprocessor CP6 (CR _m Field)	Coprocessor Register (CR _n Field)
CIDR	Core Identification Register — CIDR	0	Register 0
RCSR	Reset Cause Status Register — RCSR	1	Register 0
SINTGENR	Software Interrupt Generation Register — SINTGENR		Register 1
TARRSTR	Targeted Reset Register — TARRSTR		Register 2
	Reserved		Register 3 through 7
	Inbound MSI Interrupt Pending Registers — IMIPR [0:3]		Register 8 through 11
	Reserved		Register 12 through 15

10.4.1 Core Identification Register — CIDR

The Core Identification Register (CIDR) is a 32-bit co-processor register that provides a unique coreID number. The 81341 and 81342 supports two cores. The Transport core has a coreID equal to 0H, whereas the Application coreID is equal to 1H.

Table 447. Core Identification Register — CIDR

Co-Processor Attributes Memory Attributes		
Intel XScale® processor Local Bus Address Offset coreID0: n/a coreID1: n/a		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:03	0000 000H	Reserved.
03:00	Core Dependent	Core Identification (coreID) — This field provides a unique identification number to a core. On 81341 and 81342, there are two cores that are assigned the following coreIDs 0H and 1H. Other values are reserved.



10.4.2 Reset Cause Status Register — RCSR

The Reset Cause Status Register is a 32-bit co-processor register that indicates the cause of this core's reset. After reset, each core can read its RCSR register to figure out the cause of the reset. A core can be reset due to the following causes:

- a system reset (system reset will clear all the bits in this register)
- an internal bus caused by a core's watchdog timer reaching terminal count
- core initiated a targeted reset
- MU initiated reset
- Core initiated reset by PCI functions

Note: A system reset is indicated when all bits in this register are cleared.

Note: Some of the bits in this register are sticky bits.

Table 448. Reset Cause Status Register — RCSR

Co-Processor Attributes Memory Attributes 312824201612840 rvrvrvrvrvro		



10.4.3 Software Interrupt Generation Register — SINTGENR

The Software Interrupt Generation Register is a 32-bit co-processor register that can be used by a core to initiate an interrupt to another core in the system, including itself. This is done by specifying a coreID and an interrupt source number (ISN). The generated interrupt is posted in the Inter-Processor Interrupt Pending Register — IPIPNDR. Refer to the Interrupt Controller Unit Chapter.

Table 449. Software Interrupt Generation Register — SINTGENR

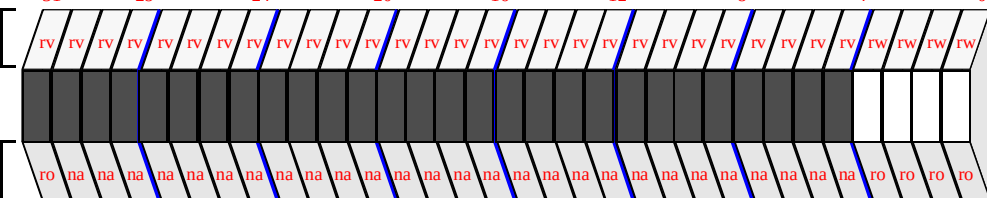
Intel XScale® processor Local Bus Address Offset coreID0 - n/a coreID1 - n/a		Intel XScale® processor Coprocessor Address CP6, CRm 1, CRn 1		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set		RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																																																			
Bit	Default	Description																																																							
31:28	0H	Reserved.																																																							
27:24	Core Dependent	coreID — This 4-bit field provides the core identification number of the core to which the interrupt is to be targeted.																																																							
23:08	0000H	Reserved.																																																							
07:05	000 ₂	Preserved.																																																							
04:00	00000 ₂	Interrupt Source Number (ISN) -- Software uses this 5-bit field to generate one of thirty-two possible software interrupts to another core in the system.81341 and 81342 only supports thirty-two interrupt source numbers.																																																							



10.4.4 Targeted Reset Register — TARRSTR

The Targeted Reset Register (TARRSTR) is a 32-bit co-processor register that can be used by a core to initiate a reset to another core in the system, including itself. Software writes this register by specifying the coreID to reset. The status of the reset is posted in the “Reset Cause Status Register — RCSR” on page 736.

Table 450. Targeted Reset Register — TARRSTR

																																															
Intel XScale® processor Local Bus Address Offset coreID0 - n/a coreID1 - n/a		Intel XScale® processor Coprocessor Address CP6, CRm 1, CRn 2														Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																																													
31:01	0000 0000H	Reserved.																																													
03:00	0000 ₂	Targeted Core Identification (coreID) — This field identifies the coreID of the core that will be the target of the reset.																																													



11.0 Interrupt Controller Unit

This chapter describes the Intel® 81341 and 81342 I/O Processors (81341 and 81342) Interrupt Controller Unit. Operation modes, setup, external memory interface, and implementation of interrupts are described in this chapter.

11.1 Overview

The interrupt control unit manages interrupt routing and interrupt sources to the Intel XScale® processor.

Interrupts are events causing a temporary break in program execution so the processor can handle another task. Interrupts commonly request I/O services or synchronize the processor with some external hardware activity. For interrupt handler portability across Intel XScale® microarchitecture family (ARM* architecture compliant), the architecture defines a consistent exception handling mechanism. To manage exceptions which include interrupt requests in parallel with processor execution, the 81341 and 81342 provides an on-chip programmable interrupt controller.

Requests for interrupt service come from many sources and are prioritized such that instruction execution is redirected only when an exception interrupt request is of higher priority than that of the executing task. On the 81341 and 81342, interrupt requests may originate from external hardware sources, internal peripherals or software. The 81341 and 81342 contains a number of integrated peripherals which may generate interrupts, including the following:

- Application DMA Channel 0, 1 and 2
- UART 0 and 1
- Peripheral Bus Interface Unit
- Performance Monitoring Unit
- ATU-E
- Inter-Processor Interrupt
- SRAM Memory Controller Unit
- I²C Bus Interface Units 0, 1 and 2
- ATU-X
- Messaging Unit
- DDR SDRAM Memory Controller Unit
- Inter-Processor Messaging Unit
- Timer 0, Timer 1 and Watch Dog Timer^a

a. Per Intel XScale® processor

The interrupt controller unit can also forward interrupts to the PCI interrupt pins (**P_INT[D:A]#**) when the PCI-X interface is being used as an endpoint. Interrupts are detected with the chip 8-bit interrupt port, an 8-bit GPIO port, and with a dedicated High-Priority Interrupt (**HPI#**) input. Interrupt requests originate from software by the **SWI** instruction. Ultimately, all interrupt sources that are steered to the Intel XScale® processor are combined into one of two internal interrupt exceptions: IRQ and FIQ.



11.2 Theory of Operation

11.2.1 Interrupt Controller Unit

The 81341 and 81342 Interrupt Controller Unit (ICU) provides the ability to generate interrupts to both the Intel XScale® processor and the PCI interrupt pins.

In addition to the internal peripherals, external devices may also generate interrupts to the Intel XScale® processor. External devices can generate interrupts via the **XINT[15:0]#** pin and the **HPI#** pin. The Interrupt Controller Unit provides the ability to direct PCI interrupts as outputs (**P_INT[D:A]#**) for the PCI-X interface when acting as an endpoint.

The Interrupt Controller Unit has two functions:

- Internal Peripheral Interrupt Control
- External Interrupt Generation

The internal peripheral interrupt control mechanism consolidates a number of interrupt sources for a given peripheral into a single interrupt driven to the Intel XScale® processor. High performance data movement associated interrupts are fully demultiplexed into the ICU, however. In order to provide the executing software with the knowledge of interrupt source, coprocessor mapped status registers describe the source of the active interrupts and the vectors to interrupt handlers for the highest priority active sources. All of the peripheral interrupts are individually enabled from the respective peripheral control registers.

In addition to the interrupts supported by the ICU, the 81341 and 81342 provides one non-maskable interrupt per Intel XScale® processor. Non-Maskable Interrupt 0 (**NMI0#**) — is driven to Intel XScale® processor 0, and Non-Maskable Interrupt 1 (**NMI1#**) — is driven to Intel XScale® processor 1. These pins are falling edge triggered input signals. **NMI0#** or **NMI1#** causes an imprecise data abort which is the second highest priority exception (higher than even an FIQ exception). Refer to [Table 451, “Exception Priorities And Vectors” on page 742](#). Since the data abort is imprecise it could potentially occur while in the middle of an abort handler, making it impossible to resume normal operation. When this an error needs to be recoverable the system should route these errors to either IRQ or FIQ by using the external interrupt input pins **XINT[0:15]#**.



11.3 The Intel XScale® Processor Exceptions Architecture

The Intel XScale® processor supports five types of exceptions²⁶, and a privileged processing mode for each type.

- IRQ and FIQ internal interrupt exceptions (normal and fast interrupts, respectively)
- memory aborts (used to implement memory protection or virtual memory)
- attempted execution of an undefined instruction
- software interrupts (SWIs) (used to make a call to an Operating System)

When an exception occurs, some of the standard registers are replaced with registers specific to the exception mode. All exceptions have replacement (or banked) registers for R14 and R13, and one interrupt mode has more registers for fast interrupt processing.

After an exception, R14 holds the return address for exception processing, which is used both to return after the exception is processed and to address the instruction that caused the exception.

R13 is banked across exception modes to provide each exception handler with a private stack pointer (SP). The fast interrupt mode also banks R8 to R12, so that interrupt processing can begin without the need to save or restore these registers. There is a seventh processing mode, System Mode, that does not have any banked registers (it uses the User mode registers), which is used to run normal (non-exception) tasks that require a privileged processor mode.

11.3.1 CPSR and SPSR

All other processor state is held in status registers. The current operating processor status is in the Current Program Status Register or CPSR. The CPSR holds:

- Four condition code flags (Negative, Zero, Carry and Overflow)
- Two interrupt disable bits (one for each type of interrupt)
- Five bits which encode the current processor mode

All five exception modes also have a Saved Program Status Register (SPSR) which holds the CPSR of the task immediately before the exception occurred. Both the CPSR and SPSR are accessed with special instructions.

26. Exception Description from the ARM Architecture Reference Manual, p. 1-3, Copyright Advanced RISC Machines Ltd. (ARM) 1996



11.3.2 The Exception Process

When an exception occurs, the Intel XScale® processor halts execution after the current instruction and begins execution at a fixed address in low memory, known as the exception vectors. There is a separate vector location for each exception (and two for memory aborts to distinguish between data and instruction accesses).

An operating system installs a handler on every exception at initialization. Privileged operating system tasks normally run in System mode to allow exceptions to occur within the operating system without state loss (exceptions overwrite their R14 when an exception occurs, and System mode is the only privileged mode that cannot be entered by an exception).

11.3.3 Exception Priorities and Vectors

It is important to note that fast interrupt (FIQ) is higher priority than the normal interrupt (IRQ). In addition, while an FIQ exception is executing, the IRQ exception is masked out.

When an exception is taken by the processor, the Program Counter (PC) is loaded with the vector associated with that exception as specified by Table 451.

Generally, the instruction at this location is required to be a branch instruction to the associated exception handler. However, in the case of an FIQ, this is not necessary since the vector location is at the very bottom of all the defined exception vectors, thus the entire FIQ exception handler can be placed at that vector location.

Table 451. Exception Priorities And Vectors

Exception	Priority	Vector ^a
Reset	1 (Highest)	0000 0000H
Data Abort	2	0000 0010H
FIQ	3	0000 001CH
IRQ	4	0000 0018H
Prefetch Abort	5	0000 000CH
Undefined Instructions	6 (Lowest)	0000 0004H
Software Interrupt (SWI) ^b	6 (Lowest)	0000 0008H

a. By enabling the Exception Vector Relocation mode (bit 13, CP15, Register 1), the Vectors (except Reset Vector) can be relocated to be based at FFFF 0000H rather than 0000 0000H. (i.e., FIQ Vector located at FFFF 001CH)

b. Undefined Instruction and SWI can not occur at the same time since SWI is a particular instruction decoding.



11.3.4 Software Requirements For Exception Handling

To use the processor's exception handling facilities, user software must provide the following items in memory:

- Exception Handler Routines
- Software handler to nest certain exceptions (i.e., FIQ and IRQ)

These items are established in memory as part of the initialization procedure.

11.3.4.1 Nesting FIQ and IRQ Exceptions

Hardware does not provide support for nesting of any particular exception, including the FIQ and IRQ exceptions.

In order to provide support for nested interrupts, a software handler must be provided to save the Link Register (R14) and the SPSR (Saved Program Status Register) before reenabling the FIQ or IRQ exception.



11.4 Intel® 81341 and 81342 I/O Processors External Interrupt Interface

The interrupt controller attached to the Intel XScale® processor has the facilities necessary to handle all core processor and peripheral internal interrupts as well as the sixteen external interrupts (**XINT[15:0]#**) and a high priority interrupt (**HPI#**).

The 81341 and 81342 Primary PCI local bus interface includes four interrupt output signals (**XINT[3:0]#/P_INT[D:A]#**). Interrupts from the Messaging Unit and Address Translation Units are routed to these interrupt output signals.

11.4.1 Interrupt Inputs

The 17 external interrupt input pins of the 81341 and 81342 have the following definitions:

- XINT[3:0]#** **External Interrupt (Input)** — These pins (**XINT[3:0]#**) cause interrupts to be requested. Each pin is a level-detect input only. These pins are internally synchronized. These pins act as interrupt inputs and must be unmasked in the INTCTL[3:0] registers. These pins also act as interrupt outputs (**P_INT[D:A]#**) for the PCI-X interface when configured as an endpoint. These pins can also function as general purpose inputs/outputs (**GPIO[11:8]**) when not used as interrupt pins.
- XINT[7:4]#** **External Interrupt (Input)** — These pins (**XINT[7:4]#**) cause interrupts to be requested. Each pin is a level-detect input only. These pins are internally synchronized. These pins only act as interrupt inputs when they are unmasked in the INTCTL[3:0] registers. These pins can also function as general purpose inputs/outputs (**GPIO[15:12]**) when not used as interrupt pins.
- XINT[15:8]#** **External Interrupt (Input)** — These pins (**XINT[15:8]#**) cause interrupts to be requested. Each pin is a level-detect input only. These pins are internally synchronized. These pins only act as interrupt inputs when they are configured as general purpose inputs and are unmasked in the INTCTL[3:0] registers. These pins can also function as general purpose inputs/outputs (**GPIO[7:0]**) when not used as interrupt pins.
- HPI#** **High-Priority Interrupt (Input)** — Causes a high priority interrupt event to occur. The external **HPI#** input requires a level input and is maskable by the INTCTL[3:0] registers. This pin is internally synchronized.
- NMIO#** **Non-Maskable Interrupt 0 (Input)** — Causes a non-maskable imprecise data abort exception to the Intel XScale® processor 0. The external **NMIO#** input requires a falling edge triggered input. This pin is internally synchronized.
- NMI1#** **Non-Maskable Interrupt 1 (Input)** — Causes a non-maskable imprecise data abort exception to the Intel XScale® processor 1. The external **NMI1#** input requires a falling edge triggered input. This pin is internally synchronized.

Note: **NMIO#** and **NMI1#** are not implemented as part of the Interrupt Controller Units. They are external pins which are falling edge triggered, internally synchronized and then directly driven to the respectively Intel XScale® processors.



The external interrupt input interface for the 81341 and 81342 consists of the pins shown in Table 452.

Table 452. Interrupt Input Pin Descriptions

Signal	Description
P_INTA# / XINT0# / GPIO[8]	This is a bi-directional pin. When 81341 and 81342 is setup as an endpoint with the PCI-X interface, this pin acts as an output pin (P_INTA#). This pin can act as an input (XINT0#) and drive the XINT0# input of the Interrupt Controller. The Interrupt Controller Unit input XINT0# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[8]) when not used as an interrupt pin.
P_INTB# / XINT1# GPIO[9]	This is a bi-directional pin. When 81341 and 81342 is setup as an endpoint with the PCI-X interface, this pin acts as an output pin (P_INTB#). This pin can act as an input (XINT1#) and drive the XINT1# input of the Interrupt Controller. The Interrupt Controller Unit input XINT1# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[9]) when not used as an interrupt pin.
P_INTC# / XINT2# / GPIO[10]	This is a bi-directional pin. When 81341 and 81342 is setup as an endpoint with the PCI-X interface, this pin acts as an output pin (P_INTC#). This pin can act as an input (XINT2#) and drive the XINT2# input of the Interrupt Controller. The Interrupt Controller Unit input XINT2# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[10]) when not used as an interrupt pin.
P_INTD# / XINT3# / GPIO[11]	This is a bi-directional pin. When 81341 and 81342 is setup as an endpoint with the PCI-X interface, this pin acts as an output pin (P_INTD#). This pin can act as an input (XINT3#) and drive the XINT3# input of the Interrupt Controller. The Interrupt Controller Unit input XINT3# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[11]) when not used as an interrupt pin.
XINT4# / GPIO[12]	This is a bi-directional pin. This pin can act as an input (XINT4#) and drive the XINT4# input of the Interrupt Controller. The Interrupt Controller Unit input XINT4# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[12]) when not used as an interrupt pin.
XINT5# / GPIO[13]	This is a bi-directional pin. This pin can act as an input (XINT5#) and drive the XINT5# input of the Interrupt Controller. The Interrupt Controller Unit input XINT5# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[13]) when not used as an interrupt pin.
XINT6# / GPIO[14]	This is a bi-directional pin. This pin can act as an input (XINT6#) and drives the XINT6# input of the Interrupt Controller. The Interrupt Controller Unit input XINT6# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL1 register. This pin can also function as a general purpose input/output pin (GPIO[14]) when not used as an interrupt pin.
XINT7# / GPIO[15]	This is a bi-directional pin. This pin can act as an input (XINT7#) and drives the XINT7# input of the Interrupt Controller. The Interrupt Controller Unit input XINT7# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. This pin can also function as a general purpose input/output pin (GPIO[15]) when not used as an interrupt pin.
XINT[15:8]# / GPIO[7:0]	These are bi-directional pins. These pins can act as inputs (XINT[15:8]#) and drive the XINT[15:8]# input of the Interrupt Controller. The Interrupt Controller Unit input XINT[15:8]# can be steered to either the FIQ or the IRQ internal interrupt input of the Intel XScale® processor. To enable a given pin as an interrupt into the ICU, it needs to be unmasked in the INTCTL[3:0] register. These pins can also function as general purpose input/output pins (GPIO[7:0]) when not used as an interrupt pin.
HPI#	HPI# input can be enabled/disabled by the INTCTL[3:0] register, but can be steered to either the FIQ or the IRQ internal interrupt.
NMI0#	NMI0# input is a non-maskable falling edge triggered signal and is internally synchronized and directly driven to Intel XScale® processor 0.
NMI1#	NMI1# input is a non-maskable falling edge triggered signal and is internally synchronized and directly driven to Intel XScale® processor 1.



11.4.2 Outbound Interrupts

When 81341 and 81342 is setup as an endpoint device with the PCI-X interface, the **XINT[3:0]#** pins act as output pins (**P_INT[D:A]#**) respectively. The Messaging Unit (MU) has the capability of generating interrupts on the PCI interrupt output pins. The MU has four distinct messaging mechanisms. Each allows a host processor or external PCI agent and the 81341 and 81342 to communicate through message passing and interrupt generation. The four mechanisms are:

- **Message Registers** — allow the 81341 and 81342 and external PCI agents to communicate by passing messages in one of four 32-bit Message Registers. In this context, a message is any 32-bit data value. Message registers combine aspects of mailbox registers and doorbell registers. Writes to the message registers may optionally cause interrupts.
- **Doorbell Registers** — allow the 81341 and 81342 to assert the PCI interrupt signals and allow external PCI agents to generate an interrupt to the Intel XScale® processor.
- **Circular Queues** — support a message passing scheme that uses four circular queues.
- **Index Registers** — support a message passing scheme that uses a portion of the 81341 and 81342 local memory to implement a large set of message registers.

All four mechanisms can result in Outbound Interrupts to a host processor.

The external interrupt output interface for 81341 and 81342 consists of the pins shown in [Table 453](#).

Table 453. Interrupt Output Pin Descriptions

Signal	Description
P_INTA#	Primary PCI Interrupt output of 81341 and 81342 source from the MU.
P_INTB#	Primary PCI Interrupt output of 81341 and 81342 source from the MU.
P_INTC#	Primary PCI Interrupt output of 81341 and 81342 source from the MU.
P_INTD#	Primary PCI Interrupt output of 81341 and 81342 source from the MU.



11.5 The Intel® 81341 and 81342 I/O Processors Interrupt Controller Unit

The 81341 and 81342 Interrupt Controller Unit (ICU) provides a flexible, low-latency means for requesting interrupts and minimizing the core's interrupt handling burden.

All interrupt sources are combined into one of the two internal interrupt exceptions: IRQ and FIQ.

The interrupt controller provides the following features for managing hardware-requested interrupts:

- Flexibility to direct interrupt sources to either the FIQ or IRQ internal interrupt exception
- 17 external interrupt pins.
 - One high-priority interrupt pin, **HPI#**.
 - Sixteen Maskable Inputs, **XINT[15:0]#**. Note that when the 81341 and 81342 acts as an endpoint with the PCI-X interface, only twelve interrupt inputs (**XINT[15:4]#**) are available instead of sixteen, as the remaining four become outputs (**P_INT[D:A]#**).
- Two internal timers sources.
- Peripheral interrupt sources.
- Two Intel XScale® processor interrupts.

All interrupts are *level sensitive*: interrupt sources must keep asserting the interrupt signal until software causes the source to deassert it.

All interrupt sources are individually maskable with the ICUs Interrupt Control registers.

Additionally, all interrupts may be quickly disabled by altering the F and I bits in the CPSR as specified in the *ARM Architecture Reference Manual*.

When software running on the 81341 and 81342 is vectored to an Interrupt Service Routine (ISR), it reads the ICUs IRQ Interrupt Vector Register (IINTVEC) or FIQ Interrupt Vector register (FINTVEC) to quickly retrieve the address for the interrupt handler of the highest priority active interrupt source.

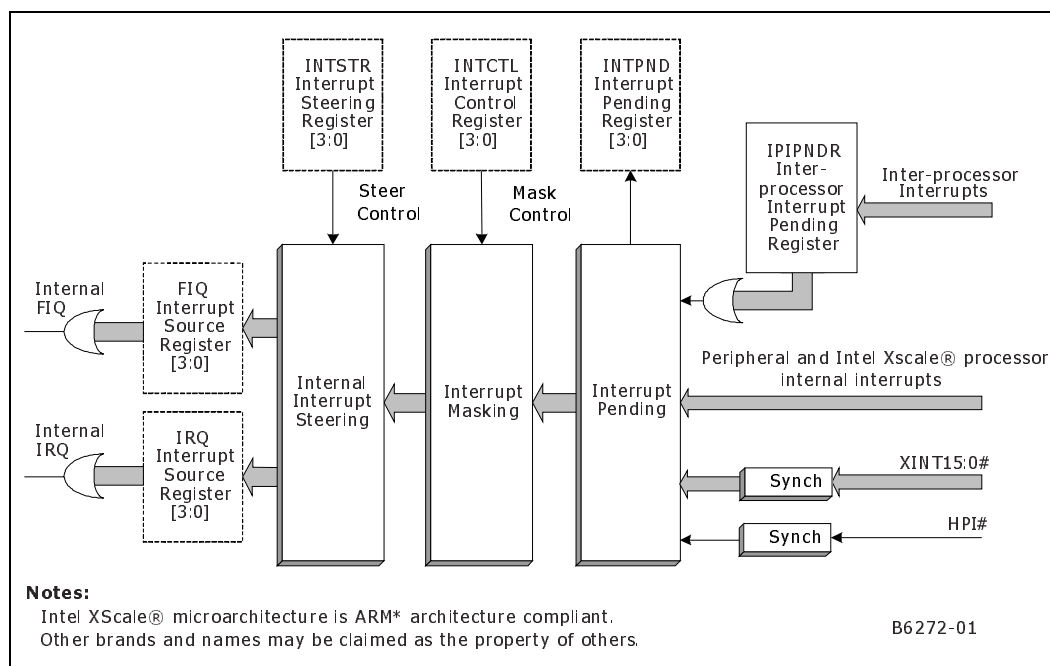
11.5.1 Programmer Model

Software has access to 15 registers in the ICU. These registers control, masking, prioritization, and vector generation for all interrupt sources.

11.5.1.1 Active Interrupt Source Control and Status

The INTCTL[3:0] are used to enable or disable (mask) individual interrupts. As mentioned, masking of all interrupts may still be accomplished via the CPSR register in the core. INTSTR[3:0] are used to direct internal interrupts to either FIQ or IRQ. IINTSRC[3:0] and FINTSRC[3:0] are read-only registers that record all currently active and unmasked interrupt sources; the architecture for the interrupt source registers and FIQ/IRQ generation is illustrated in Figure 113.

Figure 113. Interrupt Controller Block Diagram (Active Interrupt Source Registers)



11.5.1.2 Prioritization and Vector Generation for Active Interrupt Sources

IPR[7:0] registers reserve two bits for each source to assign one of four priority levels.

00 ₂ — High Priority
01 ₂ — Medium/High Priority
10 ₂ — Medium/Low Priority
11 ₂ — Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the prioritization selects a highest priority active source for each source register.

Note:

When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the highest priority active source is selected according to a fixed priority based on bit location. Highest order bit is first.



The INTBASE and INTSIZE registers are used to establish a contiguous Interrupt Service Routine (ISR) memory range for all of 128 possible sources. The architecture provides for an ISR ranging from 4 bytes to 64 Kbytes per source.

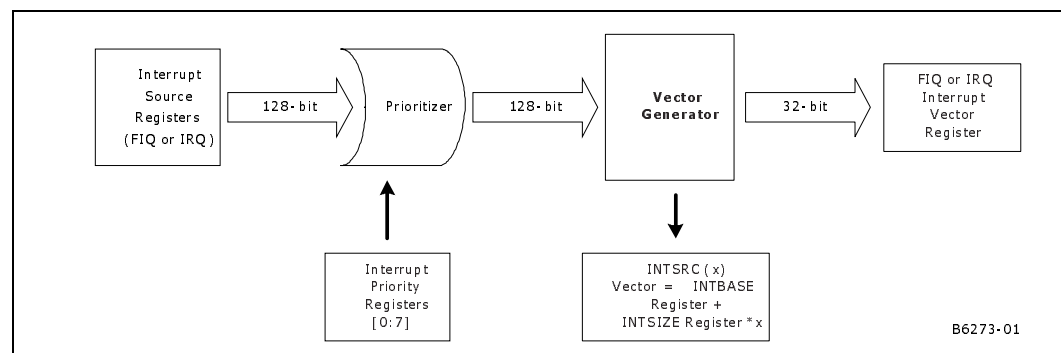
The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 128 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., IINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source) × 127 with source 127 (that is, IINTSRC3 bit 31).

Example 16. Determining the Location of the Interrupt Handler for Source 25

```
INTBASE = 0x81400000 ; 4 Mbyte Aligned Base Address
INTSIZE = 0xE ; 32 Kbytes per source (ISR Memory Range of 4 Mbytes)
ISR Address(25) = 0x81400000 + conv_hex(2^15*25) = 0x814C8000
```

Based on IINTSRC[3:0], FINTSRC[3:0], IPR[7:0], INTBASE, and INTSIZE, the interrupt controller generates the values provided by the IINTVEC and FINTVEC registers as illustrated in Figure 114. The IINTVEC and FINTVEC registers present the vector for the active interrupt source with the highest priority to the IRQ and FIQ exception handlers, respectively.

Figure 114. Interrupt Controller Block Diagram (FIQ/IRQ Interrupt Vector Generation)



Note: The 81341 and 81342 does not use all 128 possible sources.

ICU registers reside in Coprocessor 6 (CP6). They may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The instruction *CRn* field denotes the accessed register number. The instruction *opcode_1*, *opcode_2*, and *CRm* fields should be zero. Most systems restrict access to CP6 to privileged processes. To control access to CP6, use Coprocessor Access Register.

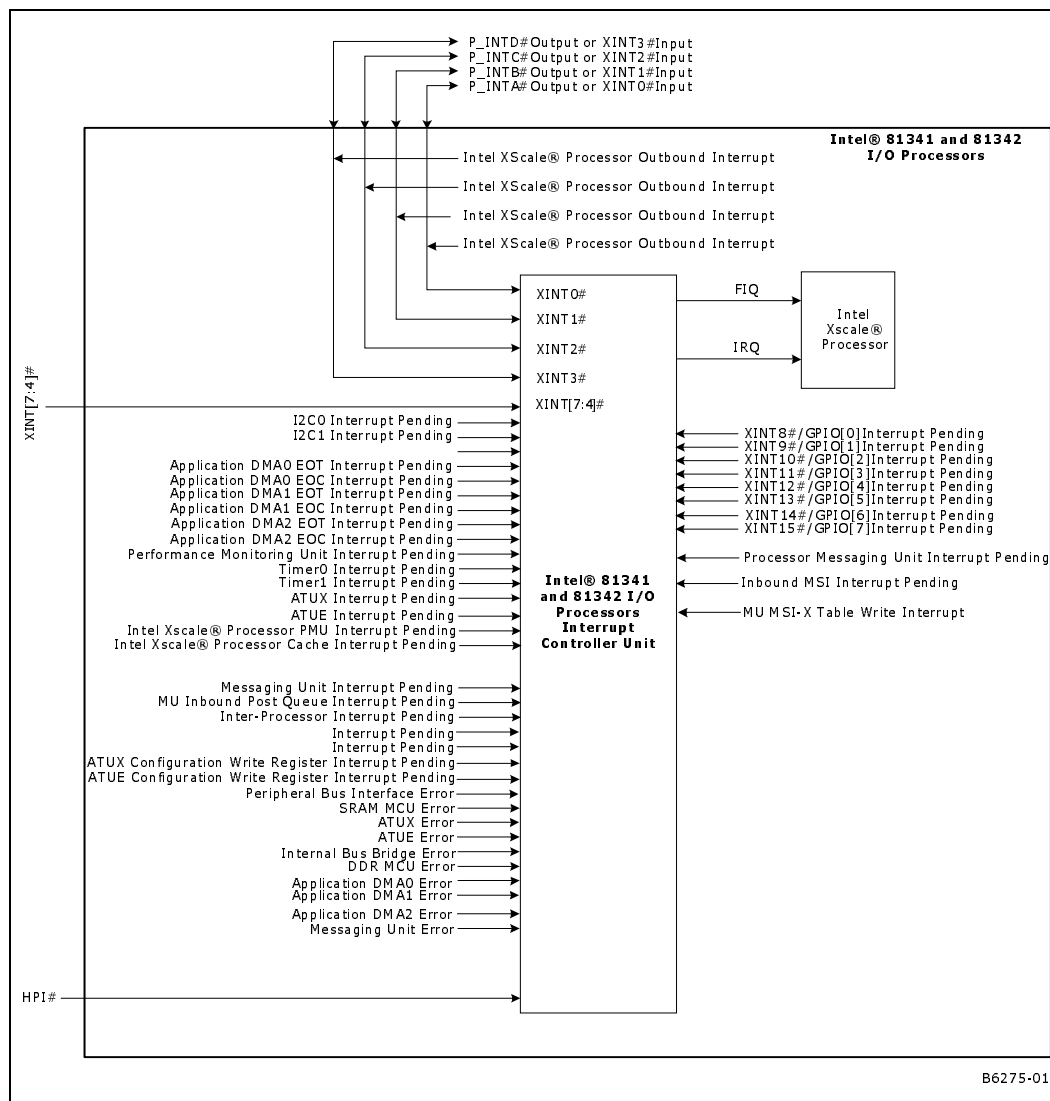
An instruction that modifies an ICU register is insured to take effect before the next instruction executes. For example, when an instruction masks an interrupt source, subsequent instructions execute in an environment in which the masked interrupt does not occur.



11.5.2 Operational Blocks

The ICU provides the connections to the Intel XScale® processor. These connections are shown in Figure 115.

Figure 115. Intel® 81341 and 81342 I/O Processors Interrupt Controller Connections





11.5.3 Intel® 81341 and 81342 I/O Processors: Internal Peripheral Interrupt

The 81341 and 81342 Interrupt Controller receives inputs from multiple internal interrupt sources. All pending interrupts required during normal operation of the various peripheral units are available in either the IINTSRC[3:0] or FINTSRC[3:0] registers depending on the value in INTSTR[3:0]. To provide the best latency for high performance event driven activities, the Application DMAs interrupts are fully demultiplexed into the interrupt source registers for FIQ and IRQ so that software does not need to access these peripheral units to diagnose the exact source and cause of the interrupt. The IINTSRC[3:0] and the FINTSRC[3:0] registers also include pending interrupts that indicate that an error has occurred in one of the peripheral units. For the interrupts that indicate errors, more detail about the exact cause of the interrupt can be determined by reading the status register of the respective peripheral unit.



11.5.3.1 Normal Interrupt Sources

The 81341 and 81342 Interrupt Controller receives normal interrupts from the Application DMA channels, Performance Monitoring Unit, the I²C Bus Interface Unit, the ATUE, the ATUX, the Programmable Timers, the Messaging Unit and the UARTs. The Application DMA channel interrupts for End of Transfer interrupt or End of Chain interrupt are demultiplexed into the interrupt controller.

A valid interrupt from any of these sources outputs a *level-sensitive* interrupt to the 81341 and 81342 Interrupt Controller input. The corresponding IRQ or FIQ interrupt source register bit in the interrupt controller should remain active as long as the interrupt is pending in the peripheral unit. The appropriate interrupt source bit is cleared by clearing the source of the interrupt at the internal peripheral.

The normal interrupt sources which drive the inputs to the 81341 and 81342 Interrupt Controller are detailed in [Table 454](#).

Note: The UART and I²C Bus Interface Unit interrupt sources are combined as a single interrupt, and include both normal and error conditions within the respective units.

Table 454. Normal Interrupt Sources

Unit	Register	Interrupt Condition
Intel XScale® processor	Overflow Flag Status Register (FLAG)	Counter Overflow
ADMA Channel 2-0	ADMA Channel Status Register 2-0	End of Chain
		End of Transfer
ATU -X & -E	ATU -X & -E Interrupt Status Register	ATU BIST Start
	Configure Reg Write	Any of the ATU Configuration registers written by an inbound Configuration Write cycle
I ² C Bus Interface Unit 2-0	I ² C Status Register 2-0	Receive Buffer Full
		Transmit Buffer Empty
		Slave Address Detect (General Call Address Detect)
		STOP Detected
		Bus Error Detected
		Arbitration Lost Detected
Messaging Unit	Inbound Interrupt Status Register	Index Register Interrupt
		Inbound Post Queue Interrupt
		Inbound Doorbell Interrupt
		Inbound Message 1 Interrupt
		Inbound Message 0 Interrupt
		MU MSI-X Table Write Interrupt
		Selective Reset Interrupt
		Coordinated Reset Interrupt
Timer 1 & 0	Timer Mode Register 1 & 0	Timer 0 has decremented to 0 interrupt.
UART Unit 1 & 0	UART 1 & 0 Interrupt ID Register	Received Line Status
		Received Data is Available
		Character Time-out Indications
		Transmit FIFO Data Request
		Autobaud Lock Indication



11.5.3.2 Error Interrupt Sources

The 81341 and 81342 Interrupt Controller receives error interrupts from the ATUs, the Messaging Unit, the Application DMA channels. Each of these interrupts represent an error condition in the peripheral unit. Refer to the appropriate units for more details.

A valid interrupt from any of these sources, outputs a *level-sensitive* interrupt to the 81341 and 81342 Interrupt Controller input. The corresponding FIQ or IRQ interrupt source register bit in the interrupt controller remains active as long as the interrupt is pending in the peripheral unit. The appropriate FIQ or IRQ interrupt source bit is cleared by clearing the source of the interrupt at the internal peripheral.

Table 455. Error Interrupt Sources

Unit	Register	Error Condition
Intel XScale® processor	L2 Cache/BIU Error Logging Register (ERRLOG)	L2 cache single bit ECC error.
ADMA Channel 2-0	Channel Status Register 2-0	IB Master Abort
		PCI Master Abort
		PCI Target Abort (master)
		Unexpected Split Completion
ATU -X & -E	ATU -X & -E Interrupt Status Register	ATU Vital Product Data Address Updated
		ATU Inbound Memory Window 1 Base Updated
		Initiated Split Completion Error Message
		Received Split Completion Error Message
		Power State Transition
		P_SERR# Asserted
		PCI Detected Parity Error
		ATU BIST Interrupt
		IB Master Abort
		P_SERR# Detected
		PCI Master Abort
		PCI Target Abort (master)
		PCI Target Abort (target)
		PCI Master Parity Error
DDR SDRAM Memory Controller	DDR Memory Controller Interrupt Status Register	ECC Error 0
		ECC Error 1
		ECC Error N
		Address Region Error
		Parity Error 0
SRAM Memory Controller	SRAM Memory Controller Interrupt Status Register	Parity Error N
		ECC Error 0
		ECC Error N
		Address Region Error
		Parity Error 0
Messaging Unit	Inbound Interrupt Status Register	Parity Error N
		Outbound Free Queue Full Interrupt
Watch Dog Time	Timer Interrupt Status Register	Error Doorbell Interrupt
		Timer Expiration



11.5.4 High-Priority Interrupt (HPI#)

The **HPI#** pin generates an interrupt for implementation of critical interrupt routines.

11.5.5 Timer Interrupts

Each of the two timer units has an associated interrupt. Timer interrupts are connected directly to the 81341 and 81342 interrupt controller and are posted in either the IINTSRC[3:0] or FINTSRC[3:0] registers. These interrupts are set up through the timer control registers described in [Chapter 12.0, “Timers.”](#)

11.5.6 Inter-Processor Interrupts

The interrupt controller provides the ability to be interrupted by another core present in the system, including self-interrupting. For example, each core can interrupt itself with this mechanism. On the 81341 and 81342 each core provides a co-processor register which software can use to initiate up to thirty-two independent interrupts to another core. The co-processor register contains a 5-bit interrupt source number (ISN), and a 4-bit coreID (CID). For example, since a CID is provided, a core can use its own CID to direct an interrupt to itself. The initiator core generates an interrupt to the targeted core by simply writing this co-processor register, and no further action is needed by the initiator core. To generate a new interrupt, the co-processor register has to be written again. By writing the co-processor register, the ISN is transferred to the targeted core interrupt controller. The receiving core's interrupt controller decodes the ISN and post an interrupt in the Inter-Processor Interrupt Pending Register (IPIPNDR). All the pending interrupts in the IPIPNDR are consolidated into a single interrupt signal Inter-Processor Interrupt Pending, and driven into the main ICU logic. Software has to clear any pending interrupts by writing 1's in the IPIPNDR.

11.5.7 Intel XScale® processor Interrupts

The Intel XScale® processor can generate two type of interrupts that are routed from the core as outputs and into the 81341 and 81342 ICU. This mechanism allows these two core interrupts to be handled like any other peripheral interrupts by the ICU. For example, these interrupts can be masked when desired using the INTCTLx registers and steered to either IRQ or FIQ using the INTSTRx registers. The Intel XScale® processor PMU interrupt is generated when the Intel XScale® processor PMU detects an overflow of one of its counters. The Intel XScale® processor Cache interrupt is generated when the Intel XScale® processor L2 cache detects a single bit ECC error. This interrupt can be used by software wishing to scrub memory when a single-bit ECC error is detected. Refer to the Intel XScale® processor External Architecture Specification for more detailed descriptions on these interrupts.

11.5.8 Software Interrupts

The application program may use the **SWI** instruction to request interrupt service.



11.6 Default Status

The interrupt logic is reset by the PCI reset signal or through software. Table 456 shows the power-up and reset values.

Table 456. Default Interrupt Routing and Status Values

Register	Default Value	Description
INTCTL0	0000 0000H	All interrupts 31:0 masked.
INTCTL1	0000 0000H	All interrupts 63:32 masked.
INTCTL2	0000 0000H	All interrupts 95:64 masked.
INTCTL3	0000 0000H	All interrupts 127:96 masked.
INTSTR0	0000 0000H	All interrupts 31:0 steered to IRQ.
INTSTR1	0000 0000H	All interrupts 63:32 steered to IRQ.
INTSTR2	0000 0000H	All interrupts 95:64 steered to IRQ.
INTSTR3	0000 0000H	All interrupts 127:96 steered to IRQ.
IINTSRC0	0000 0000H	All IRQ interrupts 31:0 inactive.
IINTSRC1	0000 0000H	All IRQ interrupts 63:32 inactive.
IINTSRC2	0000 0000H	All IRQ interrupts 95:64 inactive.
IINTSRC3	0000 0000H	All IRQ interrupts 127:96 inactive.
FINTSRC0	0000 0000H	All FIQ interrupts 31:0 inactive.
FINTSRC1	0000 0000H	All FIQ interrupts 63:32 inactive.
FINTSRC2	0000 0000H	All FIQ interrupts 95:64 inactive.
FINTSRC3	0000 0000H	All FIQ interrupts 127:96 inactive.
IPR0	0000 0000H	All interrupts 15:0 at Priority 0.
IPR1	0000 0000H	All interrupts 32:16 at Priority 0.
IPR2	0000 0000H	All interrupts 47:33 at Priority 0.
IPR3	0000 0000H	All interrupts 63:48 at Priority 0.
IPR4	0000 0000H	All interrupts 79:64 at Priority 0.
IPR5	0000 0000H	All interrupts 95:80 at Priority 0.
IPR6	0000 0000H	All interrupts 111:96 at Priority 0.
IPR7	0000 0000H	All interrupts 127:112 at Priority 0.

Note: For the default value of a register that is not listed in this table refer to the corresponding register section.



11.7 Interrupt Control Unit Registers

All Interrupt Controller registers are visible as 81341 and 81342 memory mapped registers and can be accessed through the internal memory bus. Each is a 32-bit register and is memory-mapped in the Intel XScale® processor memory space. The programmer interface to the interrupt controller is through the coprocessor registers. Table 457 describes these registers.

The coprocessor registers may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The *CR_n* field of the instruction denotes the register number to be accessed. The *opcode_1*, *opcode_2*, and *CR_m* fields of the instruction should be zero. Most systems restrict access to CP6 to privileged processes. To control access to CP6, use the Coprocessor Access Register.

Table 457. Interrupt Controller Co-Processor Register Addresses (Sheet 1 of 2)

Register Name	Description	Coprocessor CP6 (CR _m Field)	Register (CR _n Field) or MMR Address
INTBASE	Interrupt Base Register	2	Register 0
	Reserved		Register 1
INTSIZE	Interrupt Size Register		Register 2
IINTVEC	IRQ Interrupt Vector Register		Register 3
FINTVEC	FIQ Interrupt Vector Register		Register 4
IPIPNDR	Inter-Processor Interrupt Pending Register		Register 8
	Reserved		Register 9
INTPND0	Interrupt Pending Register 0	3	Register 0
INTPND1	Interrupt Pending Register 1		Register 1
INTPND2	Interrupt Pending Register 2		Register 2
INTPND3	Interrupt Pending Register 3		Register 3
INTCTL0	Interrupt Control Register 0	4	Register 0
INTCTL1	Interrupt Control Register 1		Register 1
INTCTL2	Interrupt Control Register 2		Register 2
INTCTL3	Interrupt Control Register 3		Register 3
INTSTR0	Interrupt Steering Register 0	5	Register 0
INTSTR1	Interrupt Steering Register 1		Register 1
INTSTR2	Interrupt Steering Register 2		Register 2
INTSTR3	Interrupt Steering Register 3		Register 3
IINTSRC0	IRQ Interrupt Source Register 0	6	Register 0
IINTSRC1	IRQ Interrupt Source Register 1		Register 1
IINTSRC2	IRQ Interrupt Source Register 2		Register 2
IINTSRC3	IRQ Interrupt Source Register 3		Register 3
FINTSRC0	FIQ Interrupt Source Register 0	7	Register 0
FINTSRC1	FIQ Interrupt Source Register 1		Register 1
FINTSRC2	FIQ Interrupt Source Register 2		Register 2
FINTSRC3	FIQ Interrupt Source Register 3		Register 3



Table 457. Interrupt Controller Co-Processor Register Addresses (Sheet 2 of 2)

Register Name	Description	Coprocessor CP6 (CR _m Field)	Register (CR _n Field) or MMR Address
IPR0	Interrupt Priority Register 0	8	Register 0
IPR1	Interrupt Priority Register 1		Register 1
IPR2	Interrupt Priority Register 2		Register 2
IPR3	Interrupt Priority Register 3		Register 3
IPR4	Interrupt Priority Register 4		Register 4
IPR5	Interrupt Priority Register 5		Register 5
IPR6	Interrupt Priority Register 6		Register 6
IPR7	Interrupt Priority Register 7		Register 7
		9 ^a	Register 0: 15

a. CP6 CR_m = 9H is used by the Timers.



11.7.1 Interrupt Base Register — INTBASE

The Interrupt Base Register indicates the beginning of the Interrupt Service Routine (ISR) memory range that contains the interrupt service routines for up to 128 sources. The starting address must be on a boundary equal to the granularity of the ISR memory range as specified by the INTSIZE registers.

For instance, the upper 23 bits are used for a 512 Byte ISR memory range, and the upper 16 bits for a 64 KByte ISR memory range., etc.

Table 458. Interrupt Base Register — INTBASE

Coprocessor Attributes Memory Attributes Intel XScale® processor Coprocessor address CP6, Page 2, Register 0 Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:09	0000 00H	Interrupt Base — These bits define the upper 23 bits of the base address for the ISR memory range.
08:00	00H	Reserved



11.7.2 Interrupt Size Register — INTSIZE

The Interrupt Size Register indicates the size of the Interrupt Service Routine (ISR) memory range that contains the interrupt service routines for up to 128 sources. The INTSIZE register can allocate from 4 bytes to 64 Kbytes of memory address space for the ISR per source. This means that the INTSIZE register can allocate a total ISR memory space that ranges in size from 512 bytes to 8 Mbytes.

Along with the starting address defined in the INTBASE register, the INTSIZE register fully specifies the ISR memory range.

Table 459. Interrupt Size Register — INTSIZE

Bit	Default	Description																																																			
31:04	0000000H	Reserved																																																			
03:00	0H	ISR Memory Range Size These bits define the size of the ISR memory range: <table> <tr> <th>INTSIZE</th><th>ISR Range Size</th><th>ISR Size (per Source)</th></tr> <tr> <td>0</td><td>(Disabled)</td><td></td></tr> <tr> <td>1</td><td>512 bytes</td><td>4 bytes</td></tr> <tr> <td>2</td><td>1 Kbytes</td><td>8 bytes</td></tr> <tr> <td>3</td><td>2 Kbytes</td><td>16 bytes</td></tr> <tr> <td>4</td><td>4 Kbytes</td><td>32 bytes</td></tr> <tr> <td>5</td><td>8 Kbytes</td><td>64 bytes</td></tr> <tr> <td>6</td><td>16 Kbytes</td><td>128 bytes</td></tr> <tr> <td>7</td><td>32 Kbytes</td><td>256 bytes</td></tr> <tr> <td>8</td><td>64 Kbytes</td><td>512 bytes</td></tr> <tr> <td>9</td><td>128 Kbytes</td><td>1 Kbytes</td></tr> <tr> <td>10</td><td>256 Kbytes</td><td>2 Kbytes</td></tr> <tr> <td>11</td><td>512 Kbytes</td><td>4 Kbytes</td></tr> <tr> <td>12</td><td>1 Mbytes</td><td>8 Kbytes</td></tr> <tr> <td>13</td><td>2 Mbytes</td><td>16 Kbytes</td></tr> <tr> <td>14</td><td>4 Mbytes</td><td>32 Kbytes</td></tr> <tr> <td>15</td><td>8 Mbytes</td><td>64 Kbytes</td></tr> </table>	INTSIZE	ISR Range Size	ISR Size (per Source)	0	(Disabled)		1	512 bytes	4 bytes	2	1 Kbytes	8 bytes	3	2 Kbytes	16 bytes	4	4 Kbytes	32 bytes	5	8 Kbytes	64 bytes	6	16 Kbytes	128 bytes	7	32 Kbytes	256 bytes	8	64 Kbytes	512 bytes	9	128 Kbytes	1 Kbytes	10	256 Kbytes	2 Kbytes	11	512 Kbytes	4 Kbytes	12	1 Mbytes	8 Kbytes	13	2 Mbytes	16 Kbytes	14	4 Mbytes	32 Kbytes	15	8 Mbytes	64 Kbytes
INTSIZE	ISR Range Size	ISR Size (per Source)																																																			
0	(Disabled)																																																				
1	512 bytes	4 bytes																																																			
2	1 Kbytes	8 bytes																																																			
3	2 Kbytes	16 bytes																																																			
4	4 Kbytes	32 bytes																																																			
5	8 Kbytes	64 bytes																																																			
6	16 Kbytes	128 bytes																																																			
7	32 Kbytes	256 bytes																																																			
8	64 Kbytes	512 bytes																																																			
9	128 Kbytes	1 Kbytes																																																			
10	256 Kbytes	2 Kbytes																																																			
11	512 Kbytes	4 Kbytes																																																			
12	1 Mbytes	8 Kbytes																																																			
13	2 Mbytes	16 Kbytes																																																			
14	4 Mbytes	32 Kbytes																																																			
15	8 Mbytes	64 Kbytes																																																			

11.7.3 IRQ Interrupt Vector Register — IINTVEC

The IRQ Interrupt Vector Register is a 32-bit Coprocessor 6 control register. Following an IRQ exception, the IRQ interrupt service routine reads the 32-bit vector to the ISR for the active IRQ source with the highest priority.

The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 128 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., IINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source)*127 with source 127 (that is, IINTSRC3 bit 31).

Before returning to User Mode from Interrupt Mode, the software reads the IINTVEC register and process any lower priority IRQ sources that are active. When there are no longer any active IRQ sources, a read from the IINTVEC register returns FFFF FFFFH.

Table 460. IRQ Interrupt Vector Register- IINTVEC

Intel XScale® processor Coprocessor address CP6, Page 2, Register 3 Attribute Legend: RW = Read/Write RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	IRQ Interrupt Vector—Vector to the highest priority active IRQ source. This register reads FFFF FFFFH when there are no active IRQ sources.



11.7.4 FIQ Interrupt Vector Register – FINTVEC

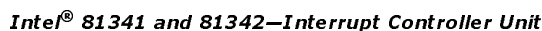
The FIQ Interrupt Vector Register is a 32-bit Coprocessor 6 control register. Following an FIQ exception, the FIQ interrupt service routine reads the 32-bit vector to the ISR for the active FIQ source with the highest priority.

The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 128 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., FINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source)*127 with source 127 (that is, FINTSRC3 bit 31).

Before returning to User Mode from Interrupt Mode, the software reads the FINTVEC register and process any lower priority FIQ sources that are active. When there are no longer any active FIQ sources, a read from the FINTVEC register returns FFFF FFFFH.

Table 461. FIQ Interrupt Vector Register- FINTVEC

Intel XScale® processor Coprocessor address CP6, Page 2, Register 4 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	FIQ Interrupt Vector—Vector to the highest priority active FIQ source. This register reads FFFF FFFFH when there are no active FIQ sources.



The Inter-Processor Interrupt Pending register is a 32-bit Coprocessor 6 control register used to specify which core generated interrupts are active. Software must write 1 to clear pending interrupts in this register. This register is located before the INTPND[3:0]. Bit 0 is set when the Interrupt Source Number (ISN) 0H message was received, bit 1 is set ISN 1H is received, and so on. Software must write 1 to clear pending interrupts in this register.

Intel XScale® processor Coprocessor address CP6, page 2, Register 8 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	0000 0000H	Inter-Processor Interrupt Pending. All the bits in this register have the same behavior. Each bit in this register is individually set based on an interrupt source number (ISN) sent by another core. For example, bit 0 is set when ISN = 00H is received, bit 1 is set when ISN = 01H, and so on. Software must write 1 to clear this bit. 0 = Not Interrupting 1 = Interrupting



11.7.6 Interrupt Pending Register 0 — INTPND0

The Interrupt Pending register 0 is a 32-bit Coprocessor 6 control register that can be used to verify pending interrupts. Software can use this registers to poll interrupts as this register is located before the INTCTL0 mask Register.

Table 463. Interrupt Pending Register 0 — INTPND0

		Intel XScale® processor Coprocessor address CP6, Page 3, Register 0																											
Bit	Default	Description																											
31	0 ₂	XINT7# Interrupt Pending.																											
30	0 ₂	XINT6# Interrupt Pending.																											
29	0 ₂	XINT5# Interrupt Pending.																											
28	0 ₂	XINT4# Interrupt Pending.																											
27	0 ₂	XINT3# Interrupt Pending.																											
26	0 ₂	XINT2# Interrupt Pending.																											
25	0 ₂	XINT1# Interrupt Pending.																											
24	0 ₂	XINT0# Interrupt Pending.																											
23:19	00H	Reserved.																											
18	0 ₂	Intel XScale® Processor Cache Interrupt Pending.																											
17	0 ₂	Intel XScale® Processor PMU Interrupt Pending.																											
16	0 ₂	Peripheral Performance Monitor Interrupt Pending.																											
15	0 ₂	ATU-X/Start BIST Interrupt Pending.																											
14	0 ₂	ATU-E Inbound Message Interrupt.																											
13	0 ₂	Messaging Unit Inbound Post Queue Interrupt Pending.																											
12	0 ₂	Messaging Unit Interrupt Pending.																											
11	0 ₂	I ² C Bus Interface 1 Interrupt Pending.																											
10	0 ₂	I ² C Bus Interface 0 Interrupt Pending.																											
9	0 ₂	Timer 1 Interrupt Pending.																											
8	0 ₂	Timer 0 Interrupt Pending.																											
7	0 ₂	Reserved.																											
6	0 ₂	Watch Dog Timer Interrupt Pending.																											
5	0 ₂	ADMA Channel 2 End-Of-Chain Interrupt Pending.																											
4	0 ₂	ADMA Channel 2 End-Of-Transfer Interrupt Pending.																											
3	0 ₂	ADMA Channel 1 End-Of-Chain Interrupt Pending.																											
2	0 ₂	ADMA Channel 1 End-Of-Transfer Interrupt Pending.																											
1	0 ₂	ADMA Channel 0 End-Of-Chain Interrupt Pending.																											
0	0 ₂	ADMA Channel 0 End-Of-Transfer Interrupt Pending.																											



11.7.7 Interrupt Pending Register 1 — INTPND1

The Interrupt Pending register 0 is a 32-bit Coprocessor 6 control register that can be used to verify pending interrupts. Software can use this registers to poll interrupts as this register is located before the INTCTL1 mask Register.

Table 464. Interrupt Pending Register 1 — INTPND1

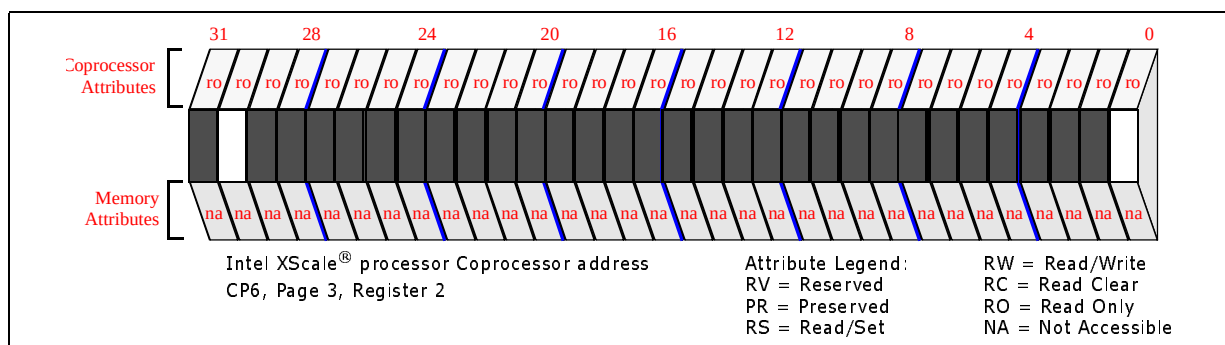
Coprocessor Attributes Memory Attributes 312824201612840 ro																															



11.7.8 Interrupt Pending Register 2 — INTPND2

The Interrupt Pending register 0 is a 32-bit Coprocessor 6 control register that can be used to verify pending interrupts. Software can use this registers to poll interrupts as this register is located before the INTCTL2 mask Register.

Table 465. Interrupt Pending Register 2 — INTPND2



Bit	Default	Description
31	0	Reserved.
30	0	South Internal Bus Bridge Error Interrupt Pending.
29	0	Reserved.
28	0	Reserved.
27	0	Reserved.
26	0	Reserved.
25	0	Reserved.
24	0	Reserved.
23	0	Reserved.
22	0	Reserved.
21	0	Reserved.
20	0	Reserved.
19	0	Reserved.
18	0	Reserved.
17	0	Reserved.
16	0	Reserved.
15	0	Reserved.
14	0	Reserved.
13	0	Reserved.
12	0	Reserved.
11	0	Reserved.
10	0	Reserved.
09	0	Reserved.
08	0	Reserved.
07	0	Reserved.
06	0	Reserved.
05	0	Reserved.
04	0	Reserved.
03	0	Reserved.
02	0	Reserved.
01	0	Reserved.
00	0	Inter-Processor Interrupt Pending.

11.7.9 Interrupt Pending Register 3 — INTPND3

The Interrupt Pending register 3 is a 32-bit Coprocessor 6 control register that can be used to verify pending interrupts. Software can use this registers to poll interrupts as this register is located before the INTCTL3 mask Register.

Table 466. Interrupt Pending Register 3 — INTPND3

Intel XScale® processor Coprocessor address
CP6, Page 3, Register 3

Attribute Legend:
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 RS = Read/Set
 RV = Reserved
 PR = Preserved
 NA = Not Accessible



11.7.10 Interrupt Control Register 0 — INTCTL0

The Interrupt Control register 0 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Table 467. Interrupt Control Register 0 — INTCTL0 (Sheet 1 of 3)

Coprocessor Attributes Memory Attributes 312824201612840 rw																															



Table 467. Interrupt Control Register 0 — INTCTL0 (Sheet 2 of 3)

Coprocessor Attributes Memory Attributes 312824201612840 rw		

**Table 467. Interrupt Control Register 0 — INTCTL0 (Sheet 3 of 3)**

		Coprocessor Attributes Memory Attributes 312824201612840 </																															



11.7.11 Interrupt Control Register 1 — INTCTL1

The Interrupt Control register 1 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Table 468. Interrupt Control Register 1 — INTCTL1 (Sheet 1 of 2)

		Intel XScale® processor Coprocessor address CP6, Page 4, Register 1 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																											
Bit	Default	Description																											
31	0 ₂	Reserved.																											
30	0 ₂	Messaging Unit Error Interrupt Mask 0 = Masked 1 = Not Masked																											
29:28	0 ₂	Reserved.																											
27	0 ₂	ADMA Channel 2 Error Interrupt Mask 0 = Masked 1 = Not Masked																											
26	0 ₂	ADMA Channel 1 Error Interrupt Mask 0 = Masked 1 = Not Masked																											
25	0 ₂	ADMA Channel 0 Error Interrupt Mask 0 = Masked 1 = Not Masked																											
24	0 ₂	Memory Controller Unit Error Interrupt Mask 0 = Masked 1 = Not Masked																											
23	0 ₂	ATU Error Interrupt Mask 0 = Masked 1 = Not Masked																											
22	0 ₂	ATU Configuration Register Write Interrupt Mask 0 = Masked 1 = Not Masked																											
21	0 ₂	Peripheral Bus Unit Error Interrupt Mask 0 = Masked 1 = Not Masked																											
20	0 ₂	UART 1 Interrupt Mask 0 = Masked 1 = Not Masked																											
19	0 ₂	UART 0 Interrupt Mask 0 = Masked 1 = Not Masked																											
18:08	000H	Reserved.																											

**Table 468. Interrupt Control Register 1 — INTCTL1 (Sheet 2 of 2)**

Intel XScale® processor Coprocessor address CP6, Page 4, Register 1 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
07	0 ₂	XINT15# Interrupt Mask. Source of this interrupt is the GPIO[7] pin. 0 = Masked 1 = Not Masked
06	0 ₂	XINT14# Interrupt Mask. Source of this interrupt is the GPIO[6] pin. 0 = Masked 1 = Not Masked
05	0 ₂	XINT13# Interrupt Mask. Source of this interrupt is the GPIO[5] pin. 0 = Masked 1 = Not Masked
04	0 ₂	XINT12# Interrupt Mask. Source of this interrupt is the GPIO[4] pin. 0 = Masked 1 = Not Masked
03	0 ₂	XINT11# Interrupt Mask. Source of this interrupt is the GPIO[3] pin. 0 = Masked 1 = Not Masked
02	0 ₂	XINT10# Interrupt Mask. Source of this interrupt is the GPIO[2] pin. 0 = Masked 1 = Not Masked
01	0 ₂	XINT9# Interrupt Mask. Source of this interrupt is the GPIO[1] pin. 0 = Masked 1 = Not Masked
00	0 ₂	XINT8# Interrupt Mask. Source of this interrupt is the GPIO[0] pin. 0 = Masked 1 = Not Masked



11.7.12 Interrupt Control Register 2 — INTCTL2

The Interrupt Control register 2 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Table 469. Interrupt Control Register 2 — INTCTL2

31

28

24

20

16

12

8

4

0

Coprocessor Attributes

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

rw

Memory Attributes

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

na

Intel XScale® processor Coprocessor address

CP6, Page 4, Register 2

Attribute Legend:

RW = Read/Write

RV = Reserved

PR = Preserved

RS = Read/Set

RC = Read Clear

RO = Read Only

NA = Not Accessible

Bit	Default	Description
31	0 ₂	Reserved.
30	0 ₂	South Internal Bus Bridge Error Interrupt Mask. 0 = Masked 1 = Not Masked
29:01	0 ₂	Reserved.
0	0 ₂	Inter-Processor Interrupt Mask 0 = Masked 1 = Not Masked



11.7.13 Interrupt Control Register 3 — INTCTL3

The Interrupt Control register 3 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Table 470. Interrupt Control Register 3 — INTCTL3 (Sheet 1 of 2)

Intel XScale® processor Coprocessor address CP6, Page 4, Register 3		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	1 ₂	HPI Interrupt Pending 0 = Masked 1 = Not Masked
30:18	0000H	Reserved.
17	0 ₂	Inbound MSI Interrupt Pending 0 = Masked 1 = Not Masked
16	0 ₂	Reserved.
15	0 ₂	MU MSI-X Table Write Interrupt Pending 0 = Masked 1 = Not Masked
14	0 ₂	ATUE Interrupt Message D Pending. 0 = Masked 1 = Not Masked
13	0 ₂	ATUE Interrupt Message C Pending. 0 = Masked 1 = Not Masked
12	0 ₂	ATUE Interrupt Message B Pending. 0 = Masked 1 = Not Masked
11	0 ₂	ATUE Interrupt Message A Pending. 0 = Masked 1 = Not Masked
10:05	0 ₂	Reserved.



Table 470. Interrupt Control Register 3 — INTCTL3 (Sheet 2 of 2)

		Intel XScale® processor Coprocessor address CP6, Page 4, Register 3 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																																			
Bit	Default	Description																																			
04	0 ₂	IMU Interrupt Pending. 0 = Masked 1 = Not Masked																																			
03	0 ₂	ATU-E Error Interrupt Pending. 0 = Masked 1 = Not Masked																																			
02	0 ₂	ATU-E Configuration Register Write Interrupt Pending. 0 = Masked 1 = Not Masked																																			
01	0 ₂	ATU-E/Start BIST Interrupt Pending. 0 = Masked 1 = Not Masked																																			
00	0 ₂	I ² C Bus Interface 2 Interrupt Pending. 0 = Masked 1 = Not Masked																																			



11.7.14 Interrupt Steering Register 0 – INTSTRO

The Interrupt Steering Register 0 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL0 register, this register steers the interrupt to an internal interrupt exception.

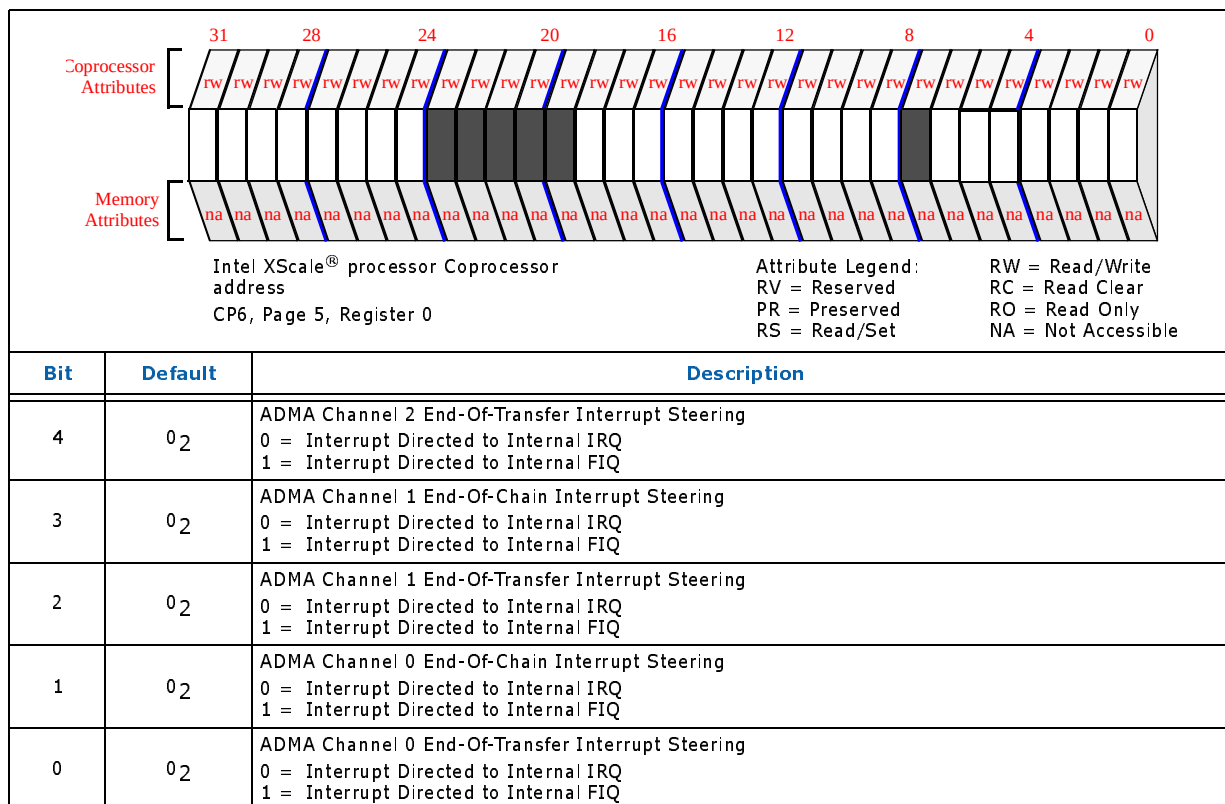
Table 471. Interrupt Steering Register 0 — INTSTR0 (Sheet 1 of 3)

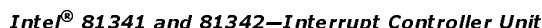
Bit	Default	Description
31	0 ₂	XINT7# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
30	0 ₂	XINT6# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
29	0 ₂	XINT5# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
28	0 ₂	XINT4# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
27	0 ₂	XINT3# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
26	0 ₂	XINT2# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
25	0 ₂	XINT1# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
24	0 ₂	XINT0# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
23:19	0 ₂	Reserved.
18	0 ₂	Intel XScale® Processor Cache Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
17	0 ₂	Intel XScale® Processor PMU Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
16	0 ₂	Peripheral Performance Monitor Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ



Table 471. Interrupt Steering Register 0 — INTSTR0 (Sheet 2 of 3)

Coprocessor Attributes Memory Attributes 312824201612840 rw		

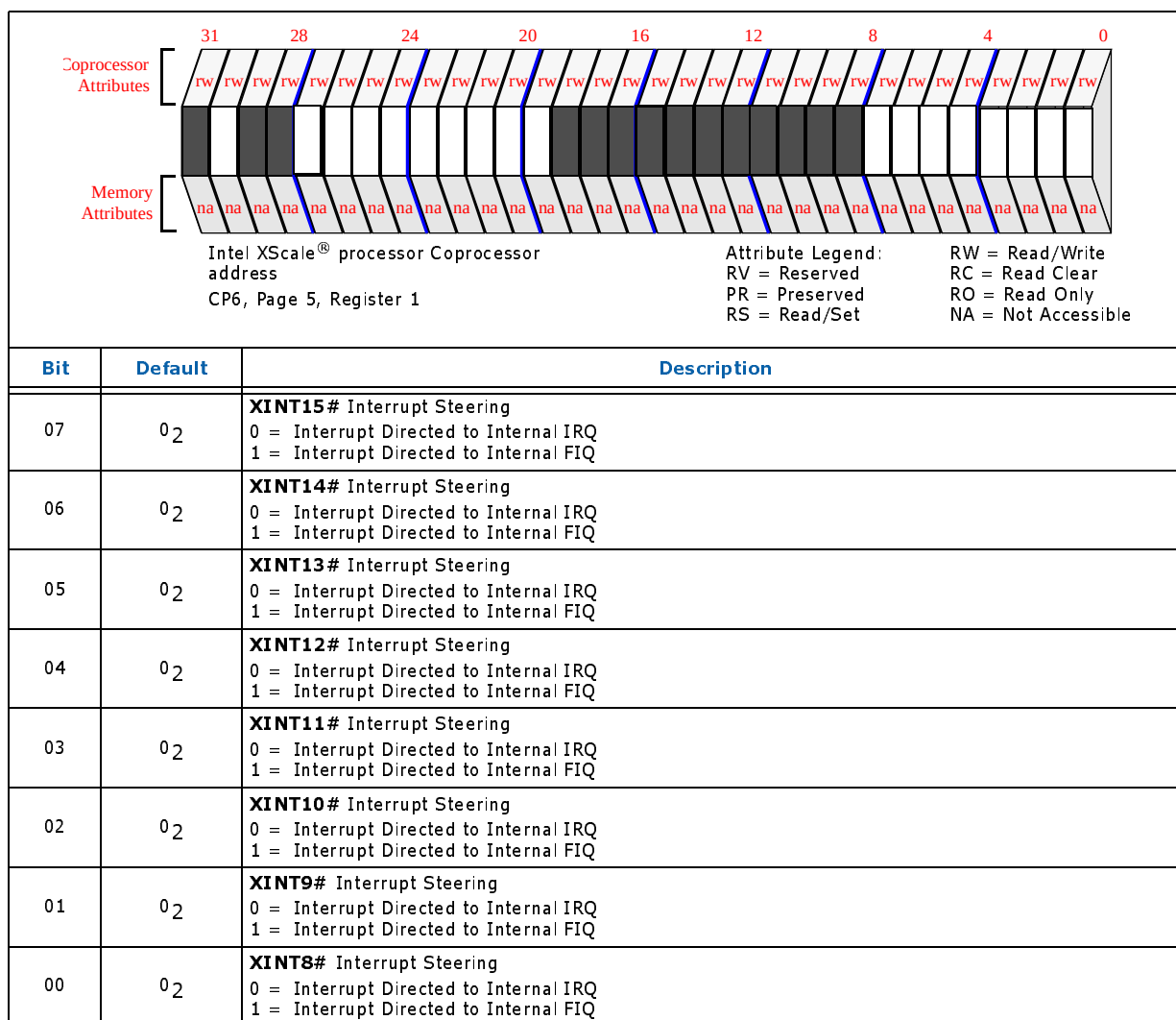
**Table 471. Interrupt Steering Register 0 – INTSTR0 (Sheet 3 of 3)**



The Interrupt Steering Register 1 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL1 register, this register steers the interrupt to an internal interrupt exception.

Bit	Default	Description
31	0 ₂	Reserved.
30	0 ₂	Messaging Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
29:28	0 ₂	Reserved.
27	0 ₂	ADMA Channel 2 Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
26	0 ₂	ADMA Channel 1 Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
25	0 ₂	ADMA Channel 0 Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
24	0 ₂	Memory Controller Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
23	0 ₂	ATU Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
22	0 ₂	ATU Configuration Register Update Interrupt 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
21	0 ₂	Peripheral Bus Interface Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
20	0 ₂	UART 1 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
19	0 ₂	UART 0 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
18:08	0000H	Reserved.

**Table 472. Interrupt Steering Register 1 – INTSTR1 (Sheet 2 of 2)**



11.7.16 Interrupt Steering Register 2 — INTSTR2

The Interrupt Steering Register 2 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL2 register, this register steers the interrupt to an internal interrupt exception.

Table 473. Interrupt Steering Register 2 — INTSTR2

Intel XScale® processor Coprocessor address
CP6, Page 5, Register 2

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

Bit	Default	Description
31	0 ₂	Reserved.
30	0 ₂	South Internal Bus Bridge Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
29:01	0 ₂	Reserved.
00	0 ₂	Inter-Processor Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ



11.7.17 Interrupt Steering Register 3 — INTSTR3

The Interrupt Steering Register 3 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL3 register, this register steers the interrupt to an internal interrupt exception.

Table 474. Interrupt Steering Register 3 — INTSTR3 (Sheet 1 of 2)

Coprocessor Attributes Memory Attributes		
Intel XScale® processor Coprocessor address CP6, Page 5, Register 3		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	HPI Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
30:18	0 ₂	Reserved.
17	0 ₂	Inbound MSI Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
16	0 ₂	Reserved.
15	0 ₂	MU MSI-X Table Write Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
14	0 ₂	ATUE Interrupt Message D Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
13	0 ₂	ATUE Interrupt Message C Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
12	0 ₂	ATUE Interrupt Message B Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
11	0 ₂	ATUE Interrupt Message A Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
10:05	0 ₂	Reserved.



Table 474. Interrupt Steering Register 3 — INTSTR3 (Sheet 2 of 2)

		Coprocessor Attributes 31 28 24 20 16 12 8 4 0 rw																																			



11.7.18 IRQ Interrupt Source Register 0 — IINTSRC0

The IRQ Interrupt Source register is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts that are steered to the internal IRQ exception are unmasked by the INTCTL0 register and active. The INTSTRO control register is used to steer individual interrupts to the IRQ exception.

The IINTSRC0 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an IRQ interrupt.

Table 475. IRQ Interrupt Source Register 0 — IINTSRC0 (Sheet 1 of 3)

Intel XScale® processor Coprocessor address CP6, page 6, Register 0		
Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set		
RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:	0 ₂	XINT7# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
30:	0 ₂	XINT6# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
29:	0 ₂	XINT5# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
28:	0 ₂	XINT4# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
27:	0 ₂	XINT3# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
26:	0 ₂	XINT2# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
25:	0 ₂	XINT1# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
24:	0 ₂	XINT0# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
23:19	0 ₂	Reserved.
18:	0 ₂	Intel XScale® Processor Cache Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
17:	0 ₂	Intel XScale® Processor PMU Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0



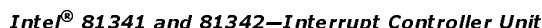
Table 475. IRQ Interrupt Source Register 0 — IINTSRC0 (Sheet 2 of 3)

Coprocessor Attributes 312824201612840 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro		Memory Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na	
Intel XScale® processor Coprocessor address CP6, page 6, Register 0		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	

Bit	Default	Description
16	0 ₂	Peripheral Performance Monitor Interrupt — when set, at least one of the programmable event counters and/or the Global Time Stamp Counter contains an overflow condition. Application software identifies the counter by reading the Event Monitoring Interrupt Status register (EMISR). 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
15	0 ₂	ATU/Start BIST Interrupt — when set, the host processor has set the start BIST request in the ATUBISTR register. 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
14	0 ₂	ATU-E Inbound Message Interrupt — when set, the ATU has set the Inbound Vendor Message Received bit in the ATUISR register. 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
13	0 ₂	Messaging Unit Inbound Post Queue Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
12	0 ₂	Messaging Unit Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
11	0 ₂	I ² C Bus Interface 1 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
10	0 ₂	I ² C Bus Interface 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
9	0 ₂	Timer 1 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
8	0 ₂	Timer 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
7	0 ₂	Reserved.
6	0 ₂	Watch Dog Timer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
5	0 ₂	ADMA Channel 2 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0

**Table 475. IRQ Interrupt Source Register 0 — IINTSRC0 (Sheet 3 of 3)**

Intel XScale® processor Coprocessor address CP6, page 6, Register 0 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
4	0 ₂	ADMA Channel 2 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
3	0 ₂	ADMA Channel 1 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
2	0 ₂	ADMA Channel 1 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
1	0 ₂	ADMA Channel 0 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
0	0 ₂	ADMA Channel 0 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0



The IRQ Interrupt Source register is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts that are steered to the internal IRQ exception are unmasked by the INTCTL1 register and active. The INTSTR1 control register is used to steer individual interrupts to the IRQ exception.

Table 476. IRQ Interrupt Source Register 1 — IINTSRC1 (Sheet 1 of 2)

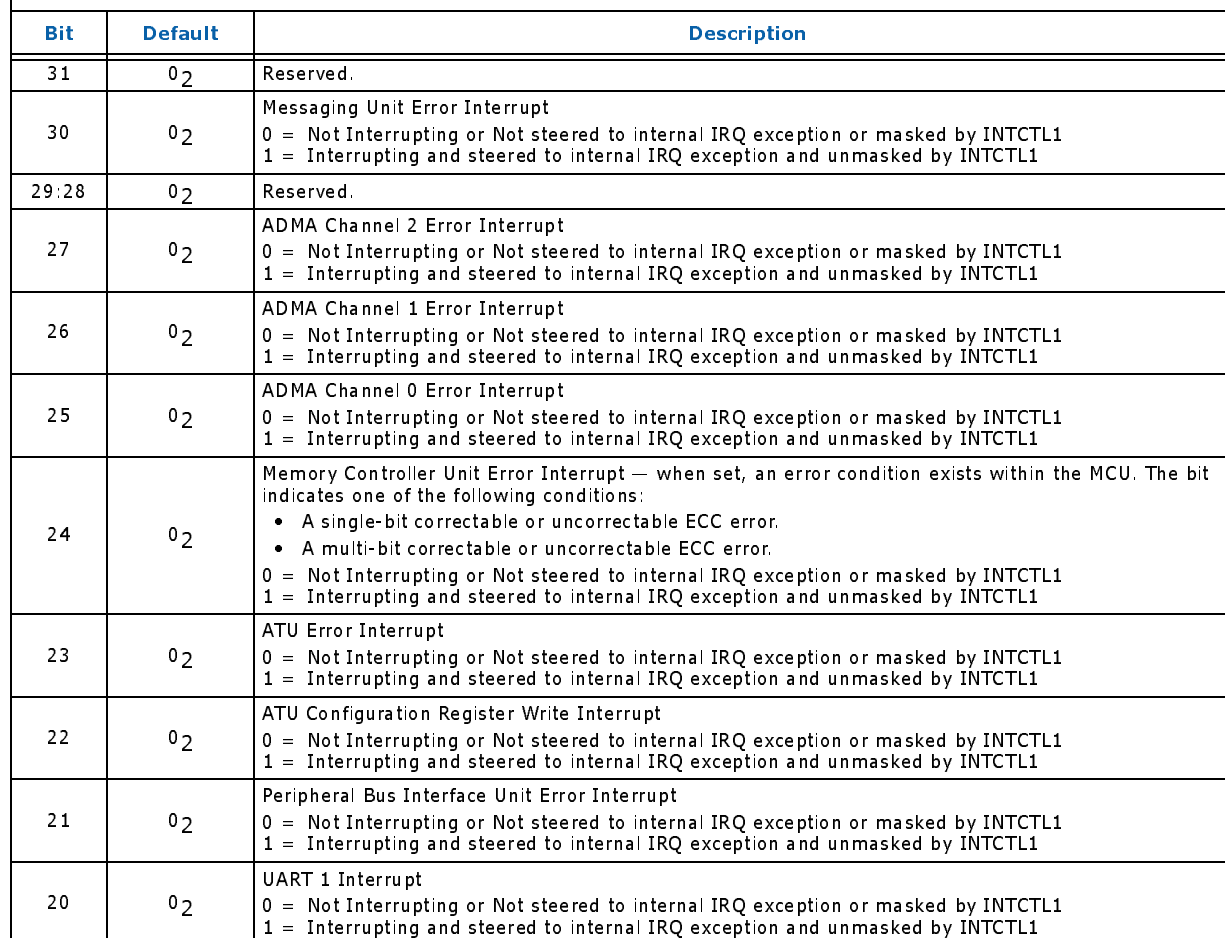


**Table 476. IRQ Interrupt Source Register 1 — IINTSRC1 (Sheet 2 of 2)**

Coprocessor Attributes 312824201612840 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Memory Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor Coprocessor address CP6, Page 6, Register 1 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																													
19	0 ₂	UART 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
18:08	0000H	Reserved.																													
7	0 ₂	XINT15# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
6	0 ₂	XINT14# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
5	0 ₂	XINT13# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
4	0 ₂	XINT12# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
3	0 ₂	XINT11# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
2	0 ₂	XINT10# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
1	0 ₂	XINT9# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													
0	0 ₂	XINT8# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1																													



11.7.21 IRQ Interrupt Source Register 3 — IINTSRC3

The IRQ Interrupt Source register is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts that are steered to the internal IRQ exception are unmasked by the INTCTL3 register and active. The INTSTR3 control register is used to steer individual interrupts to the IRQ exception.

The IINTSRC3 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an IRQ interrupt.

Table 478. IRQ Interrupt Source Register 3 — IINTSRC3 (Sheet 1 of 2)

Coprocessor Attributes Memory Attributes		
Intel XScale® processor Coprocessor address CP6, Page 6, Register 3		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	HPI Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
30:18	0000 0H	Reserved.
17	0 ₂	Inbound MSI Interrupt 0 = Not interrupting or not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
16	0 ₂	Reserved.
15	0 ₂	MU MSI-X Table Write Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
14	0 ₂	ATUE Interrupt Message D 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
13	0 ₂	ATUE Interrupt Message C 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
12	0 ₂	ATUE Interrupt Message B 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
11	0 ₂	ATUE Interrupt Message A 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3
10:05	0 ₂	Reserved.



Table 478. IRQ Interrupt Source Register 3 — IINTSRC3 (Sheet 2 of 2)

		Coprocessor Attributes 31 28 24 20 16 12 8 4 0 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Memory Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor Coprocessor address CP6, Page 6, Register 3 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																																			
Bit	Default	Description																																			
04	0 ₂	IMU Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3																																			
03	0 ₂	ATU-E Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3																																			
02	0 ₂	ATU-E Configuration Register Write Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3																																			
01	0 ₂	ATU-E/Start BIST Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3																																			
00	0 ₂	I ² C Bus Interface 2 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL3 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL3																																			

11.7.22 FIQ Interrupt Source Register 0 — FINTSRC0

The FIQ Interrupt Source register 0 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL0 register and active. The INTSTR0 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC0 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Table 479. FIQ Interrupt Source Register 0 — FINTSRC0 (Sheet 1 of 3)

Bit	Default	Description
31	0 ₂	XINT7# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
30:	0 ₂	XINT6# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
29:	0 ₂	XINT5# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
28:	0 ₂	XINT4# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
27	0 ₂	XINT3# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
26	0 ₂	XINT2# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
25	0 ₂	XINT1# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
24	0 ₂	XINT0# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
23:19	0 ₂	Reserved.
18	0 ₂	Intel XScale® Processor Cache Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
17	0 ₂	Intel XScale® Processor PMU Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0



Table 479. FIQ Interrupt Source Register 0 — FINTSRC0 (Sheet 2 of 3)

Coprocessor Attributes 31 28 24 20 16 12 8 4 0 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro		Memory Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na	
Intel XScale® processor Coprocessor address CP6, Page 7, Register 0		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	

Bit	Default	Description
16	0 ₂	Peripheral Performance Monitor Interrupt — when set, at least one of the programmable event counters and/or the Global Time Stamp Counter contains an overflow condition. Application software identifies the counter by reading the Event Monitoring Interrupt Status register (EMISR). 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
15	0 ₂	ATU/Start BIST Interrupt — when set, the host processor has set the start BIST request in the ATUBISTR register. 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
14	0 ₂	ATU-E Inbound Message Interrupt — when set, the ATU has set the Inbound Vendor Message Received bit in the ATUISR register. 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
13	0 ₂	Messaging Unit Inbound Post Queue Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
12	0 ₂	Messaging Unit Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
11	0 ₂	I ² C Bus Interface 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
10	0 ₂	I ² C Bus Interface 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
9	0 ₂	Timer 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
8	0 ₂	Timer 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
7	0 ₂	Reserved.
6	0 ₂	Watch Dog Timer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
5	0 ₂	ADMA Channel 2 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0

**Table 479. FIQ Interrupt Source Register 0 — FINTSRC0 (Sheet 3 of 3)**

Intel XScale® processor Coprocessor address CP6, Page 7, Register 0 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
4	0 ₂	ADMA Channel 2 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
3	0 ₂	ADMA Channel 1 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
2	0 ₂	ADMA Channel 1 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
1	0 ₂	ADMA Channel 0 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
0	0 ₂	ADMA Channel 0 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0



11.7.23 FIQ Interrupt Source Register 1 — FINTSRC1

The FIQ Interrupt Source register 1 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL1 register and active. The INTSTR1 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC1 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Table 480. FIQ Interrupt Source Register 1 — FINTSRC1 (Sheet 1 of 2)

Coprocessor Attributes Memory Attributes		
Intel XScale® processor Coprocessor address CP6, Page 7, Register 1		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	Reserved.
30	0 ₂	Messaging Unit Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
29:28	0 ₂	Reserved.
27	0 ₂	ADMA Channel 2 Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
26	0 ₂	ADMA Channel 1 Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
25	0 ₂	ADMA Channel 0 Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
24	0 ₂	Memory Controller Unit Error Interrupt — when set, an error condition exists within the MCU. The bit indicates one of the following conditions: - A single-bit correctable or uncorrectable ECC error. - A multi-bit correctable or uncorrectable ECC error. 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
23	0 ₂	ATU Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
22	0 ₂	ATU Configuration Register Write Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
21	0 ₂	Peripheral Bus Interface Unit Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
20	0 ₂	UART 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1

**Table 480. FIQ Interrupt Source Register 1 — FINTSRC1 (Sheet 2 of 2)**

Coprocessor Attributes 312824201612840 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro Memory Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor Coprocessor address CP6, Page 7, Register 1 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																													
19	0 ₂	UART 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
18:08	0000H	Reserved.																													
7	0 ₂	XINT15# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
6	0 ₂	XINT14# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
5	0 ₂	XINT13# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
4	0 ₂	XINT12# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
3	0 ₂	XINT11# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
2	0 ₂	XINT10# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
1	0 ₂	XINT9# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													
0	0 ₂	XINT8# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1																													



11.7.24 FIQ Interrupt Source Register 2 — FINTSRC2

The FIQ Interrupt Source register 2 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL2 register and active. The INTSTR2 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC2 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Table 481. FIQ Interrupt Source Register 2 — FINTSRC2

		Intel XScale® processor Coprocessor address CP6, Page 7, Register 2 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																											
Bit	Default	Description																											
31	0 ₂	Reserved.																											
30	0 ₂	South Internal Bus Bridge Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL2 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL2																											
29:01	0 ₂	Reserved.																											
00	0 ₂	Inter-Processor Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL2 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL2																											



11.7.25 FIQ Interrupt Source Register 3 — FINTSRC3

The FIQ Interrupt Source register 3 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL3 register and active. The INTSTR3 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC3 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Table 482. FIQ Interrupt Source Register 3 — FINTSRC3 (Sheet 1 of 2)

Coprocessor Attributes Memory Attributes		
Intel XScale® processor Coprocessor address CP6, Page 7, Register 3		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set
		RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31	0 ₂	HPI Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
30:18	0 ₂	Reserved.
17	0 ₂	Inbound MSI Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
16	0 ₂	Reserved.
15	0 ₂	MU MSI-X Table Write Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
14	0 ₂	ATUE Interrupt Message D 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
13	0 ₂	ATUE Interrupt Message C 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
12	0 ₂	ATUE Interrupt Message B 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
11	0 ₂	ATUE Interrupt Message A 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3
10:05	0 ₂	Reserved.



Table 482. FIQ Interrupt Source Register 3 — FINTSRC3 (Sheet 2 of 2)

Bit	Default	Description																													
04	0 ₂	IMU Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3																													
03	0 ₂	ATU-E Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3																													
02	0 ₂	ATU-E Configuration Register Write Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3																													
01	0 ₂	ATU-E/Start BIST Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3																													
00	0 ₂	I²C Bus Interface 2 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL3 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL3																													



11.7.26 Interrupt Priority Register 0 — IPR0

The Interrupt Priority Register 0 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 15 down to 0. The IPR0 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers:

- 00₂ — High Priority
- 01₂ — Medium/High Priority
- 10₂ — Medium/Low Priority
- 11₂ — Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the highest priority vectors pending for either FIQ or IRQ are presented in the FINTVEC or IINTVEC respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 483. Interrupt Priority Register 0 — IPR0

		Intel XScale® processor Coprocessor address CP6, page 8, Register 0																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set								RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible							
Bit	Default	Description																															
31:30	00 ₂	ATU/Start BIST Interrupt Priority																															
29:28	00 ₂	ATU-E Inbound Message Interrupt Priority																															
27:26	00 ₂	Messaging Unit Inbound Post Queue Interrupt Priority																															
25:24	00 ₂	Messaging Unit Interrupt Priority																															
23:22	00 ₂	I ² C Bus Interface 1 Interrupt Priority																															
21:20	00 ₂	I ² C Bus Interface 0 Interrupt Priority																															
19:18	00 ₂	Timer 1 Interrupt Priority																															
17:16	00 ₂	Timer 0 Interrupt Priority																															
15:14	00 ₂	Reserved.																															
13:12	00 ₂	Watch Dog Timer Interrupt Priority																															
11:10	00 ₂	ADMA Channel 2 End-Of-Chain Interrupt Priority																															
09:08	00 ₂	ADMA Channel 2 End-Of-Transfer Interrupt Priority																															
07:06	00 ₂	ADMA Channel 1 End-Of-Chain Interrupt Priority																															
05:04	00 ₂	ADMA Channel 1 End-Of-Transfer Interrupt Priority																															
03:02	00 ₂	ADMA Channel 0 End-Of-Chain Interrupt Priority																															
01:00	00 ₂	ADMA Channel 0 End-Of-Transfer Interrupt Priority																															



11.7.27 Interrupt Priority Register 1 — IPR1

The Interrupt Priority Register 1 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 31 down to 15. The IPR1 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers:

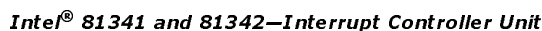
- 00₂ — High Priority
- 01₂ — Medium/High Priority
- 10₂ — Medium/Low Priority
- 11₂ — Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the highest priority vectors pending for either FIQ or IRQ are presented in the FINTVEC or IINTVEC respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 484. Interrupt Priority Register 1 — IPR1

Coprorocessor Attributes Memory Attributes		
Intel XScale® processor Coprocessor address CP6, Page 8, Register 1		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:30	00 ₂	XINT7# Interrupt Priority
29:28	00 ₂	XINT6# Interrupt Priority
27:26	00 ₂	XINT5# Interrupt Priority
25:24	00 ₂	XINT4# Interrupt Priority
23:22	00 ₂	XINT3# Interrupt Priority
21:20	00 ₂	XINT2# Interrupt Priority
19:18	00 ₂	XINT1# Interrupt Priority
17:16	00 ₂	XINT0# Interrupt Priority
15:06	0000H	Reserved.
05:04	00 ₂	Intel XScale® Processor Cache Interrupt Priority
03:02	00 ₂	Intel XScale® Processor PMU Interrupt Priority
01:00	00 ₂	Peripheral Performance Monitor Interrupt Priority



The Interrupt Priority Register 3 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 63 down to 48. The IPR3 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers;

00₂ – High Priority
01₂ – Medium/High Priority
10₂ – Medium/Low Priority
11₂ – Low Priority

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Intel XScale® processor Coprocessor address
CP6, Page 8, Register 3

Attribute Legend:
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RW = Read/Write
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible

11.7.30 Interrupt Priority Register 4 — IPR4

The Interrupt Priority Register 4 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 79 down to 64. The IPR4 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers:

00₂ – High Priority

01₂ – Medium/High Priority

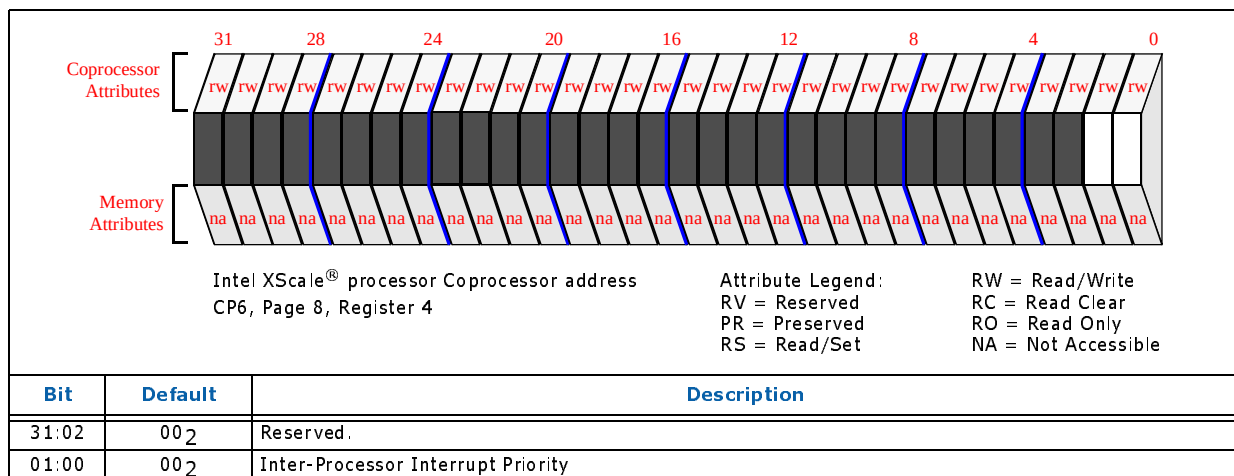
10₂ – Medium/Low Priority

11₂ – Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the highest priority vectors pending for either FIQ or IRQ are presented in the FINTVEC or IINTVEC respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 487. Interrupt Priority Register 4 — IPR4





11.7.31 Interrupt Priority Register 5 — IPR5

The Interrupt Priority Register 5 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 95 down to 80. The IPR5 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers:

- 00₂ — High Priority
- 01₂ — Medium/High Priority
- 10₂ — Medium/Low Priority
- 11₂ — Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the highest priority vectors pending for either FIQ or IRQ are presented in the FINTVEC or IINTVEC respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 488. Interrupt Priority Register 5 — IPR5

Intel XScale® processor Coprocessor address CP6, Page 8, Register 5 Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																													
31:30	00 ₂	Reserved.																													
29:28	00 ₂	South Internal Bus Bridge Error Interrupt Priority.																													
27:00	00 ₂	Reserved.																													



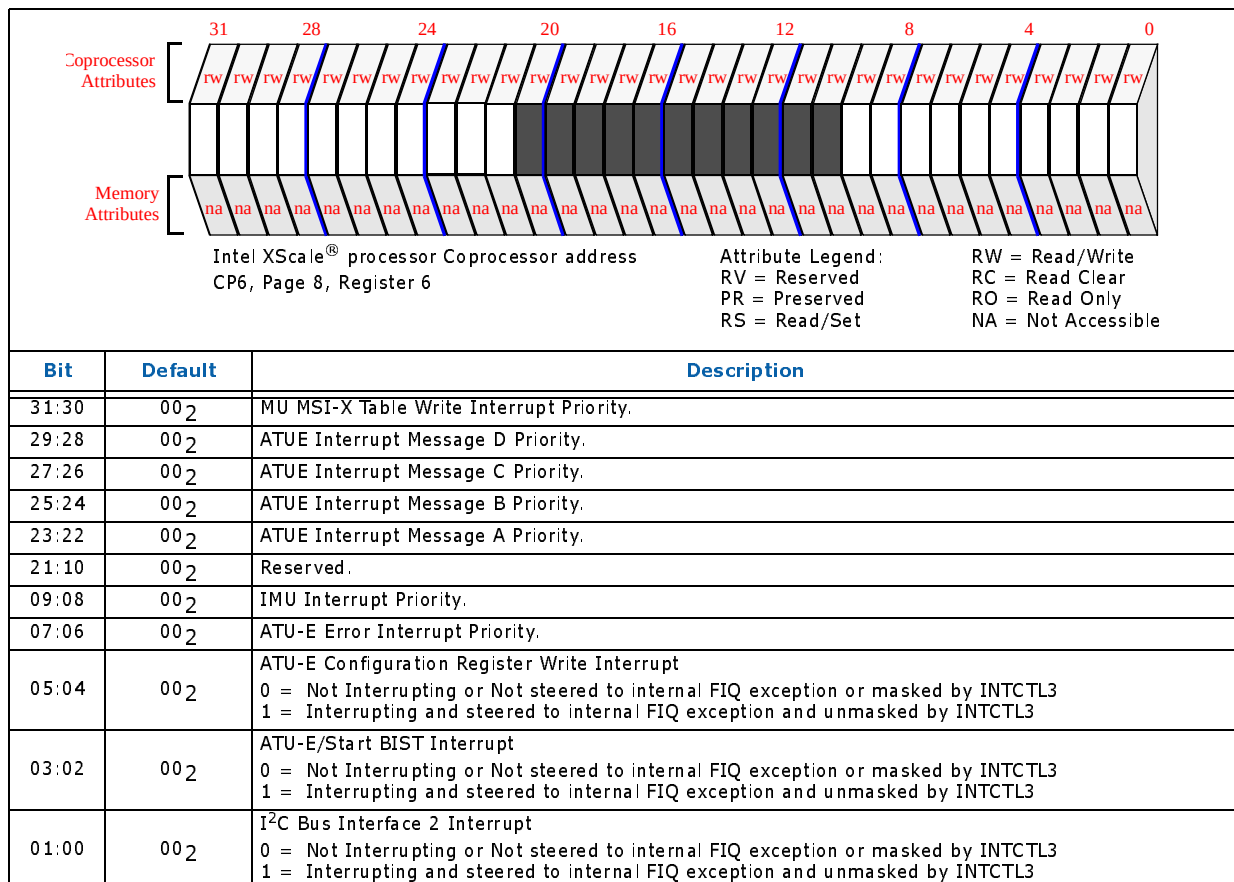
11.7.32 Interrupt Priority Register 6 – IPR6

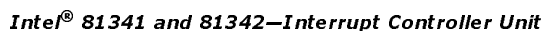
The Interrupt Priority Register 6 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 111 down to 96. The IPR6 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers:

- 00₂ – High Priority
01₂ – Medium/High Priority
10₂ – Medium/Low Priority
11₂ – Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[3:0] or the IINTSRC[3:0] registers, the highest priority vectors pending for either FIQ or IRQ are presented in the FINTVEC or IINTVEC respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 489. Interrupt Priority Register 6 — IPR6



The Interrupt Priority Register 7 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 127 down to 112. The IPR7 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[3:0] registers;

00₂ – High Priority
01₂ – Medium/High Priority
10₂ – Medium/Low Priority
11₂ – Low Priority

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

[illegible]

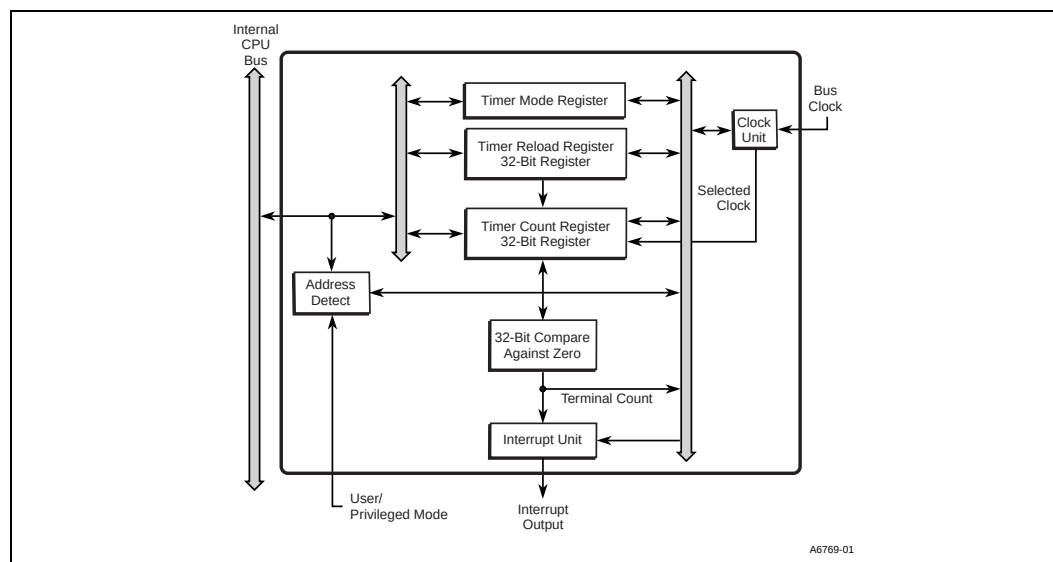
12.0 Timers

This chapter describes the Intel XScale® processor dual-programmable 32-bit timers and Watch Dog Timer. Topics include timer registers (TMRx, TCRx and TRRx), timer operation, timer interrupts, and timer register values at initialization.

Each timer is programmed by the timer registers. These registers are mapped into Intel XScale® processor Coprocessor 6, registers 0 to 8. They may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The *CRn* field of the instruction denotes the register number to be accessed. The *opcode_1* and *opcode_2* fields of the instruction should be zero. The *CRm* field of the instruction should be nine. Most systems restrict access to CP6 to privileged processes. To control access to CP6, use the Coprocessor Access Register.

Figure 116 shows a diagram of the timer functions. See also Figure 117 for the Programmable Timer state diagram.

Figure 116. Programmable Timer Functional Diagram



When enabled, a timer decrements the user-defined count value with each Timer Clock (TCLOCK) cycle. The countdown rate is also user-configurable to be equal to the internal bus frequency, or the internal bus clock rate divided by 4, 8 or 16. The timers can be programmed to either stop when the count value reaches zero (single-shot mode) or run continuously (auto-reload mode). When a timer's count reaches zero, the timer's interrupt unit signals the processor's interrupt controller.

Table 491. Timer Performance Ranges

Internal Bus Frequency (MHz)	Max Resolution (ns)	Max Range (mins)
400	2.5	2.86



12.1 Timer Operation

This section summarizes the programmable timer and Watch Dog Timer operation and describes load/store access latency for the timer registers.

12.1.1 Basic Programmable Timer Operation

Each timer has a programmable enable bit in its control register (TMRx.enable) to start and stop counting. This allows the programmer to prevent user mode tasks from enabling or disabling the timer. Once the timer is enabled, the value stored in the Timer Count Register (TCRx) decrements every Timer Clock (TCLOCK) cycle. TCLOCK is determined by the Timer Input Clock Select (TMRx.csel) bit setting. The countdown rate can be set to equal the internal bus clock frequency, or the internal bus clock rate divided by 4, 8 or 16. Setting TCLOCK to a slower rate lets the user specify a longer count period with the same 32-bit TCRx value.

Software can read or write the TCRx value whether the timer is running or stopped. This lets the user monitor the count without using hardware interrupts.

When the TCRx value decrements to zero, the unit's interrupt request signals the processor's interrupt controller. See [Section 12.2, "Timer Interrupts" on page 811](#) for more information. The timer checks the value of the timer reload bit (TMRx.reload) setting. When TMRx.reload = 1, the processor:

- Automatically reloads TCRx with the value in the Timer Reload Register (TRRx).
- Decrements TCRx until it equals 0 again.

This process repeats until software clears TMRx.reload or TMR.enable.

When TMRx.reload = 0, the timer stops running and sets the terminal count bit (TMRx.tc). This bit remains set until user software reads or writes the TMRx register. Either access type clears the bit. The timer ignores any value specified for TMRx.tc in a write request.

Table 492. Timer Mode Register Control Bit Summary

TRRx	TCRx	Bit 2 (TMRx.reload)	Bit 1 (TMRx.enable)	Action
X	X	X	0	Timer disabled.

Note: X = don't care
N = a number between 1H and FFFF FFFFH



12.1.2 Watch Dog Timer Operation

The Watch Dog Timer (WDT) is a 32-bit down counter that can be used to reset the Internal Bus and the Intel XScale® processor or generate an interrupt when software gets stuck in an infinite loop. Refer to [Section 502, “Watch Dog Timer Setup Register — WDTSR” on page 819](#) for setting up the Watchdog timer. A reset of the Internal Bus also results in the **M_RST#** output to be asserted which can be used to reset the system or as an external indicator of the WDT expiration.

Following **P_RST#** assertion, the WDT is disabled.

The software can enable the WDT by using coprocessor instructions (that is, MCR or LDC) to write the value 1E1E 1E1EH followed by the value E1E1 E1E1H to the WDT Control register. When enabled, the WDT is initialized with FFFF FFFFH and begin to decrement towards 0000 0000H.

The software is required periodically to write the WDT initialization sequence (the value 1E1E 1E1EH followed by the value E1E1 E1E1H) to the WDT Control register in order to reset the timer value to FFFF FFFFH. For a 300 MHz internal bus, this means that the sequence must be written approximately every fourteen seconds.

Note: The WDT always runs at Intel XScale® processor speed without any prescaling.

When the software fails to reinitialize the WDT prior to the timer value transitioning to zero, an Internal Bus Reset is generated. This reinitializes all Internal Bus peripherals and the Intel XScale® processor.

Once enabled, the WDT can be disabled by writing the value 1F1F 1F1FH followed by the value F1F1 F1F1H to the WDT Control register. The WDT can be enabled again by writing the value 1E1E 1E1EH followed by the value E1E1 E1E1H to the WDT Control register.

12.1.3 Load/Store Access Latency for Timer Registers

As with all other load accesses from internal memory-mapped registers, a load instruction that accesses a timer register has a latency of one internal processor cycle. With one exception, a store access to a timer register completes and all state changes take effect before the next instruction begins execution. The exception to this is when disabling a timer. Latency associated with the disabling action is such that a timer interrupt may be posted immediately after the disabling instruction completes. This can occur when the timer is near zero as the store to TMRx occurs. In this case, the timer interrupt is posted immediately after the store to TMRx completes and before the next instruction can execute. [Table 493](#) summarizes the timer access and response timings. Refer also to the individual register descriptions for details.

Note that the processor may delay the actual issuing of the load or store operation due to previous instruction activity and resource availability of processor functional units.

The processor ensures that the TMRx.tc bit is cleared within one internal bus clock after a load or store instruction accesses TMRx.

Table 493. Timer Responses to Register Bit Settings

Name	Status	Action
(TMRx.tc) Terminal Count Bit 0	READ	Timer clears this bit when user software accesses TMRx. This bit can be set 1 internal bus clock later. The timer sets this bit within 1 internal bus clock of TCRx reaching zero when TMRx.reload=0.
	WRITE	Timer clears this bit within 1 internal bus clock after the software accesses TMRx. The timer ignores any value specified for TMRx.tc in a write request.
(TMRx.enable) Timer Enable Bit 1	READ	Bit is available 1 internal bus clock after executing a read instruction from TMRx.
	WRITE	Writing a '1' enables the internal bus clock to decrement TCRx within 1 internal bus clock after executing a store instruction to TMRx.
(TMRx.reload) Timer Auto Reload Enable Bit 2	READ	Bit is available 1 internal bus clock after executing a read instruction from TMRx.
	WRITE	Writing a '1' enables the reload capability within 1 internal bus clock after the store instruction to TMRx has executed. The timer loads TRRx data into TCRx and decrements this value during the next internal bus clock cycle.
(TMRx.csel1:0) Timer Input Clock Select Bits 4-5	READ	Bits are available 1 internal bus clock after executing a read instruction from TMRx.csel1:0 bit(s).
	WRITE	The timer re-synchronizes the clock cycle used to decrement TCRx within one internal bus clock cycle after executing a store instruction to TMRx.csel1:0 bit(s).
(TCRx.d31:0) Timer Count Register	READ	The current TCRx count value is available within 1 internal bus clock cycle after executing a read instruction from TCRx. When the timer is running, the pre-decremented value is returned as the current value. When the timer is transferring the TRRx count into TCRx in the current count cycle, the timer returns the new TCRx count value to the executing read instruction.
	WRITE	The value written to TCRx becomes the active value within 1 internal bus clock cycle. When the timer is running, the value written is decremented in the current clock cycle.
(TRRx.d31:0) Timer Reload Register	READ	The current TRRx count value is available within 1 internal bus clock after executing a read instruction from TRRx.
	WRITE	The value written to TRRx becomes the active value stored in TRRx within 1 internal bus clock cycle. When the timer is transferring the TRRx value into the TCRx, data written to TRRx is also transferred into TCRx.



12.2 Timer Interrupts

Each timer is the source for one interrupt. When a timer detects a zero count in its TCRx, the timer generates an internal level-detected Timer Interrupt signal (TINTx) to the interrupt controller, and the interrupt source (INTSRC[1:0]) bit is set in the interrupt controller. Each timer interrupt can be selectively masked in the Interrupt Control (INTCTL[1:0]) registers. Refer to the Interrupt Controller Unit Chapter for a description of interrupt controller operation.

After servicing the timer interrupt, the interrupt service routine clears the pending request by writing a '1' to the appropriate bit of the Timer Interrupt Status Register (TISR).

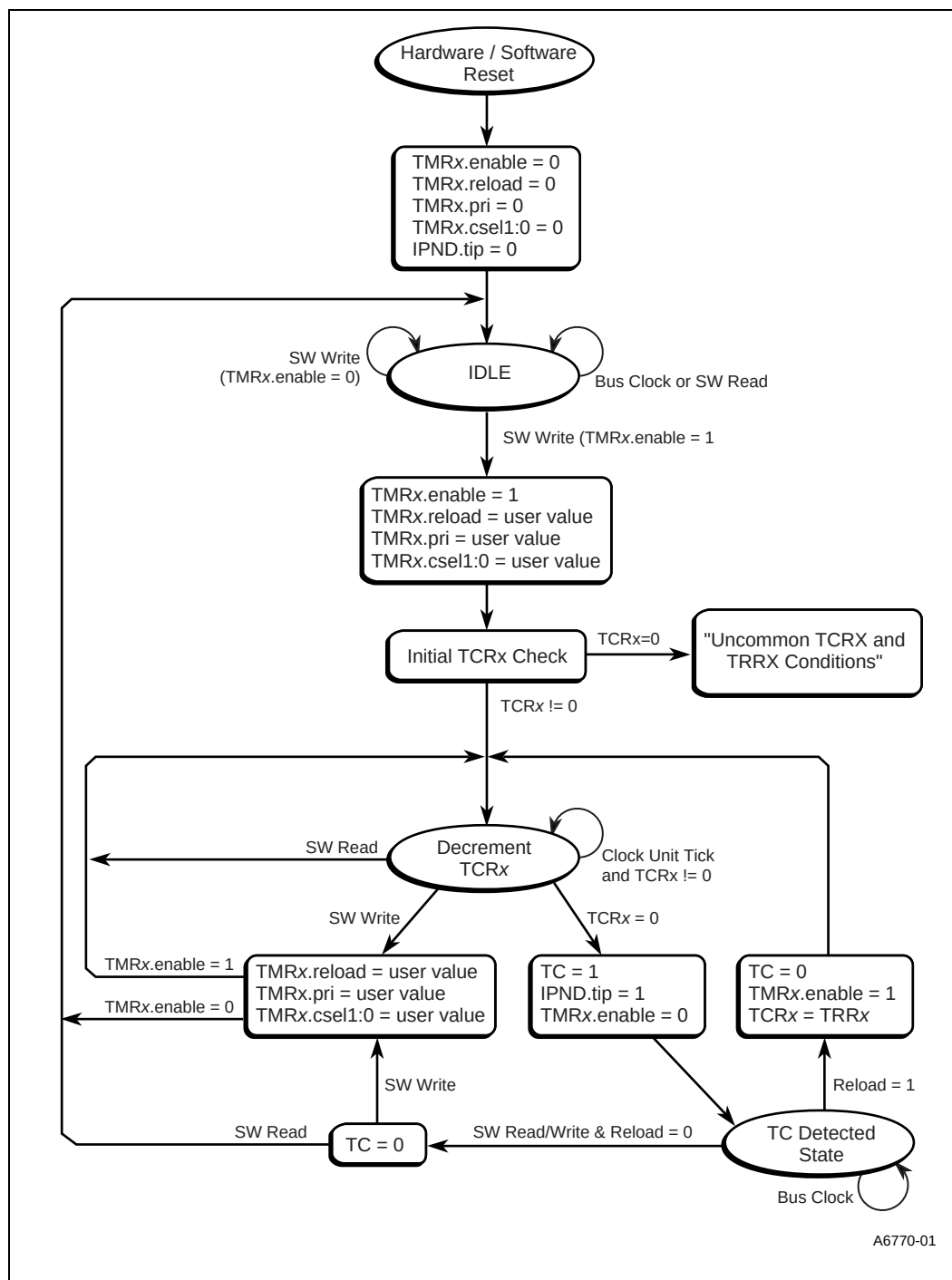
When a timer generates a second interrupt request before the CPU services the first interrupt request, the second request may be lost.

When auto-reload is enabled for a timer, the timer continues to decrement the value in TCRx even after entry into the timer interrupt handler.

12.3 Timer State Diagram

Figure 117 shows the common states of the Timer Unit. For uncommon conditions see Section 12.5, “Uncommon TCRx and TRRx Conditions” on page 820.

Figure 117. Timer Unit State Diagram





12.4 Timer Registers

As shown in Table 494, each timer has three co-processor registers:

- Timer Mode Register — programs the specific mode of operation or indicates the current programmed status of the timer. This register is described in Section 12.4.2, “Timer Mode Registers – TMR0:1” on page 814.
- Timer Count Register — contains the timer’s current count. See Section 12.4.3, “Timer Count Register – TCR0:1” on page 817.
- Timer Reload Register — contains the timer’s reload count. See Section 12.4.4, “Timer Reload Register – TRR0:1” on page 817.

Table 494. Timer Registers

Timer Unit	Register Acronym	Register Name
Timer 0	TMR0	Timer Mode Register 0
	TCR0	Timer Count Register 0
	TRR0	Timer Reload Register 0
Timer 1	TMR1	Timer Mode Register 1
	TCR1	Timer Count Register 1
	TRR1	Timer Reload Register 1

12.4.1 Power Up/Reset Initialization

Upon assertion of P_RST#, the timer registers are initialized to the values shown in Table 495.

Table 495. Timer Power Up Mode Settings

Mode/Control Bit	Notes
TMRx.tc = 0	No terminal count
TMRx.enable = 0	Prevents counting and assertion of TINTx
TMRx.reload = 0	Single terminal count mode
TMRx.pri = 0	Privileged Mode and User Mode Writes Allowed
TMRx.csel1:0 = 0	Timer Clock = internal bus clock
TCRx.d31:0 = 0	Undefined
TRRx.d31:0 = 0	Undefined
TINTx output	Deasserted



12.4.2 Timer Mode Registers – TMR0:1

The Timer Mode Register (TMRx) lets the user program the mode of operation and determine the current status of the timer. TMRx bits are described in the subsections following [Table 496](#) and are summarized in [Table 492](#).

Table 496. Timer Mode Register – TMRx

Intel XScale® processor Coprocessor address TMR0: CP6, Page 9, Register 0 TMR1: CP6, Page 9, Register 1 Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
31:06	0000 000H	Reserved. Initialize to 0.
05:04	00 ₂	Timer Input Clock Selects — TMRx.csel1:0 (00) 1:1 Timer Clock = internal bus clock (01) 4:1 Timer Clock = internal bus clock / 4 (10) 8:1 Timer Clock = internal bus clock / 8 (11) 16:1 Timer Clock = internal bus clock / 16
03	0 ₂	Timer Register Privileged Write Control — TMRx.pri (0) Privileged and User Mode Write Enabled (1) Privileged Mode Only Write Enabled
02	0 ₂	Timer Auto Reload Enable — TMRx.reload (0) Auto Reload Disabled (1) Auto Reload Enabled
01	0 ₂	Timer Enable — TMRx.enable (0) Disabled (1) Enabled
00	0 ₂	Terminal Count Status — TMRx.tc (0) No Terminal Count (1) Terminal Count



12.4.2.1 Bit 0 — Terminal Count Status Bit (TMRx.tc)

The TMRx.tc bit is set when the Timer Count Register (TCRx) decrements to 0 and bit 2 (TMRx.reload) is not set for a timer. The TMRx.tc bit allows applications to monitor timer status through software instead of interrupts. TMRx.tc remains set until software accesses (reads or writes) TMRx. The access clears TMRx.tc. The timer ignores any value specified for TMRx.tc in a write request.

When auto-reload is selected for a timer and the timer is enabled, the TMRx.tc bit status is unpredictable. Software should not rely on the value of the TMRx.tc bit when auto-reload is enabled.

The processor also clears the TMRx.tc bit upon hardware or software reset. Refer to [Section 19.2, “Reset Overview” on page 993](#).

12.4.2.2 Bit 1 — Timer Enable (TMRx.enable)

TMRx.enable bit allows user software to control the timer's RUN/STOP status. When:

TMRx.enable = 1 The Timer Count Register (TCRx) value decrements every Timer Clock (TCLOCK) cycle. TCLOCK is determined by the Timer Input Clock Select (TMRx.csel bits 0-1). See [Section 12.4.2.5](#). When TMRx.reload=0, the timer automatically clears TMRx.enable when the count reaches zero. When TMRx.reload=1, the bit remains set. See [Section 12.4.2.3](#).

TMRx.enable = 0 The timer is disabled and ignores all input transitions.

User software sets this bit. Once started, the timer continues to run, regardless of other processor activity. Three events can stop the timer:

- User software explicitly clearing this bit (i.e., TMRx.enable = 0).
- TCRx value decrements to 0, and Timer Auto Reload Enable (TMRx.reload) bit = 0.
- Hardware or software reset. Refer to [Section 19.2, “Reset Overview” on page 993](#).

12.4.2.3 Bit 2 — Timer Auto Reload Enable (TMRx.reload)

The TMRx.reload bit determines whether the timer runs continuously or in single-shot mode. When TCRx = 0 and TMRx.enable = 1 and:

TMRx.reload = 1 The timer runs continuously. The processor:

1. Automatically loads TCRx with the value in the Timer Reload Register (TRRx), when TCRx value decrements to 0.
2. Decrements TCRx until it equals 0 again.

Steps 1 and 2 repeat until software clears TMRx bits 1 or 2.

TMRx.reload = 0 The timer runs until the Timer Count Register = 0. TRRx has no effect on the timer.

User software sets this bit. When TMRx.enable and TMRx.reload are set and TRRx does not equal 0, the timer continues to run in auto-reload mode, regardless of other processor activity. Two events can stop the timer:

- User software explicitly clearing either TMRx.enable or TMRx.reload.
- Hardware or software reset.

The processor clears this bit upon hardware or software reset.



12.4.2.4 Bit 3 — Timer Register Privileged Read/Write Control (TMRx.pri)

The TMRx.pri bit enables or disables user mode writes to the timer registers (TMRx, TCRx, TRRx). Privileged mode writes are allowed regardless of this bit's condition. Software can read these registers from either mode. Note that TMR1.pri also controls write access to the "Watch Dog Timer Control Register — WDTCR" on page 819 and the "Watch Dog Timer Setup Register — WDTSR" on page 819.

When:

- TMRx.pri = 1 The timer ignores the user mode write to the timer registers; however, writes from the privileged modes are allowed.
- TMRx.pri = 0 The timer registers can be written from either the user mode or the privileged modes.

The processor clears TMRx.pri upon hardware or software reset.

12.4.2.5 Bits 4, 5 — Timer Input Clock Select (TMRx.csel1:0)

User software programs the TMRx.csel bits to select the Timer Clock (TCLOCK) frequency. See Table 497. As shown in Figure 116, the internal bus clock is an input to the timer clock unit. These bits allow the application to specify whether TCLOCK runs at or slower than the internal bus clock frequency.

Table 497. Timer Input Clock (TCLOCK) Frequency Selection

Bit 5 TMRx.csel1	Bit 4 TMRx.csel0	Timer Clock (TCLOCK)
0	0	Timer Clock = internal bus clock
0	1	Timer Clock = internal bus clock / 4
1	0	Timer Clock = internal bus clock / 8
1	1	Timer Clock = internal bus clock / 16

The processor clears these bits upon hardware or software reset (TCLOCK = Core Clock).



12.4.3 Timer Count Register – TCR0:1

The Timer Count Register (TCRx) is a 32-bit register that contains the timer's current count. The register value decrements with each timer clock tick. When this register value decrements to zero (terminal count), a timer interrupt is generated. When TMRx.reload is not set for the timer, the status bit in the timer mode register (TMRx.tc) is set and remains set until the TMRx register is accessed. Table 498 shows the timer count register.

Table 498. Timer Count Register – TCRx

CP { 31 rw rw rw rw rw rw rw rw 28 rw rw rw rw rw rw rw rw 24 rw rw rw rw rw rw rw rw 20 rw rw rw rw rw rw rw rw 16 rw rw rw rw rw rw rw rw 12 rw rw rw rw rw rw rw rw 8 rw rw rw rw rw rw rw rw 4 rw rw rw rw rw rw rw rw 0 rw rw rw rw rw rw rw rw } MMR { na																															

The valid programmable range is from 1H to FFFF FFFFH. Avoid programming TCRx to 0 as it has varying results as described in Section 12.5, “Uncommon TCRx and TRRx Conditions” on page 820. User software can read or write TCRx whether the timer is running or stopped. Bit 3 of TMRx determines user read/write control (Section 12.4.2.5). The TCRx value is undefined after hardware or software reset.

12.4.4 Timer Reload Register – TRR0:1

The Timer Reload Register (TRRx; Table 499) is a 32-bit register that contains the timer's reload count. The timer loads the reload count value into TCRx when TMRx.reload is set (1), TMRx.enable is set (1) and TCRx equals zero.

As with TCRx, the valid programmable range is from 1H to FFFF FFFFH. Avoid programming a value of 0, as it may prevent TINTx from asserting continuously. (See Section 12.5, “Uncommon TCRx and TRRx Conditions” on page 820 for more information.)

User software can access TRRx whether the timer is running or stopped. Bit 3 of TMRx determines read/write control (Section 12.4.2.5, “Bits 4, 5 — Timer Input Clock Select (TMRx.csel1:0)” on page 816). TRRx value is undefined after hardware or software reset.

Table 499. Timer Reload Register – TRRx

CP [31 28 24 20 16 12 8 4 0 rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw MMR [na na na na na na na na na na na na na na na na na na na na na na na na na na na na																															
Intel XScale® processor																Attribute Legend:															
Coprocessor address																RV = Reserved															
TRR0: CP6, Page 9, Register 4																PR = Preserved															
TRR1: CP6, Page 9, Register 5																RS = Read/Set															
																RW = Read/Write															
																RC = Read Clear															
																RO = Read Only															
																NA = Not Accessible															
31:00	0000 0000H					Timer Auto-Reload Value — TRRx.d31:0																									



Timer Interrupt Status Register – TISR

The Timer Interrupt Status Register (TISR; [Table 500](#)) is a three-bit register that contains the timer's pending interrupt status and the Watchdog pending interrupt status (when enabled). The setting of these status bits represents the assertion of a "level-sensitive" interrupt request to the Interrupt Controller Unit. After the interrupt service routine completes processing of the interrupt request, it needs to write a '1' to the appropriate bit in the TISR to clear the pending request.

TISR interrupt requests are cleared after hardware or software reset.

Table 500. Timer Interrupt Status Register — TISR

Intel XScale® processor
Coprocessor address
TISR: CP6, Page 9, Register 6

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible



12.4.6 Watch Dog Timer Control Register – WDTCR

The Watch Dog Timer Control Register (WDTCR) is a 32-bit register that software can use to enable the WDT or read the current WDT count value. The register value decrements with each internal bus clock tick. When this register value decrements to zero (terminal count), an Internal Bus Reset or an interrupt is generated. Refer to [Section 502, “Watch Dog Timer Setup Register – WDTSR” on page 819](#). The timer can be enabled and/or reinitialized by writing 1E1E 1E1EH immediately followed by E1E1 E1E1H to the WDTCR. Once enabled, the WDT can be disabled by writing 1F1F 1F1FH immediately followed by F1F1 F1F1H to the WDTCR.

Note: This register is also controlled by the TMR1.pri (TMR1[3]) bit. For example, it can be written only by privileged processes.

Table 501. Watch Dog Timer Control Register – WDTCR

Intel XScale® processor Coprocessor address WDTCR: CP6, Page 9, Register 7		
Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
31:00	0000 0000H	Watch Dog Timer Count Value — By writing 1E1E 1E1EH followed by E1E1 E1E1H to this register, software can enable the WDT and reset the count value to FFFF FFFFH. When read, this register returns the current value contained in the WDT. Once enabled, the WDT can be disabled by writing 1F1F 1F1FH immediately followed by F1F1 F1F1H to the WDTCR

12.4.7 Watch Dog Timer Setup Register – WDTSR

The Watch Dog Timer Setup Register (WDTSR) is a 32-bit register that software can use to select the action taken when the WDT register value decrements to zero (terminal count) — either an interrupt or an internal bus reset can be generated.

Note: This register is also controlled by the TMR1.pri (TMR1[3]) bit. For example, it can be written only by privileged processes.

Table 502. Watch Dog Timer Setup Register – WDTSR

Intel XScale® processor Coprocessor address WDTSR: CP6, Page 9, Register 8		
Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
31	0 ₂	Reserved.
30:01	0000 000H	Reserved.
00	0 ₂	Watch Dog Timer Event Selector bit — This bit selects whether the WDT generates an interrupt or an internal bus reset when the WDT register value decrements to zero (terminal count). When an interrupt is chosen, the interrupt is posted in the Timer Interrupt Status Register (TISR). Refer to Section 500, “Timer Interrupt Status Register – TISR” on page 818 . 0 = An interrupt is generated 1 = A reset is generated



12.5 Uncommon TCRx and TRRx Conditions

Table 492 summarizes the most common settings for programming the timer registers. Under certain conditions, however, it may be useful to set the Timer Count Register or the Timer Reload Register to zero before enabling the timer. Table 503 details the conditions and results when these conditions are set.

Table 503. Uncommon TMRx Control Bit Settings

TRRx	TCRx	Bit 2 (TMRx.reload)	Bit 1 (TMRx.enable)	Action
X	0	0	1	TMRx.tc and TINTx set, TMR.enable cleared
0	0	1	1	Timer and auto reload enabled, TINTx not generated and timer enable remains set.
0	N	1	1	Timer and auto reload enabled. TINTx set when TCRx=0. The timer remains enabled but further TINTx's are not generated.

Note: X = don't care
N = a number between 1H and FFFF FFFFH



13.0 Inter-Processor Messaging Unit

This chapter describes the Inter-processor Messaging Unit (IMU). IMU operation modes, setup, external interface, and implementation are detailed in this chapter.

13.1 Overview

The inter-processor messaging unit provides a method for the two integrated Intel XScale® processor processors in to efficiently communicate.

The hardware provides two communication mechanisms:

- Doorbell Registers
- Circular Queues

The IMUs Circular Queue mechanism consists of eight queues. Four of the queues are for queueing messages from Processor 0 to Processor 1 while the other four are for queueing messages from Processor 1 to Processor 0. From the point of view of a particular processor, there are four Send queues and four Receive queues.

Attention bits are provided that can interrupt a given Intel XScale® processor processor when queue full/empty conditions occur as appropriate.

For both the Doorbell Register and Circular Queue mechanism, the IMU provides a fixed priority scheme. When multiple Doorbell Control Register bits are set, an interrupt service routine can use this hardware to process Doorbell or Circular queue mechanism needs in a fixed priority order.



13.2 Door Bell Registers

The Door Bell Registers provide the ability for an integrated Intel XScale® processor to send up to 15 individual doorbells to another integrated Intel XScale® processor.

When any of a given Intel XScale® processor's doorbell status (DBSTAT) bits are set by the other Intel XScale® processor (see [Section 13.6.1, "Door Bell Control Register — DBCR" on page 832](#)), an interrupt (when enabled) is signaled to that Intel XScale® processor.

The IMU provides individual interrupt enables for each Door Bell bit (see [Section 13.6.2, "Door Bell Enable Register — DBER" on page 834](#)) and the ability to read the individual interrupt enables for the door bell bits of the other processor (see [Section 13.6.4, "Door Bell Enable Other Processor Register — DBEOR" on page 835](#)).

Writing the Door Bell Assertion Register ([Section 13.6.3, "Door Bell Assertion Register — DBAR" on page 835](#)) will set the Door Bell status bits on the other processor.

The Door Bell Registers are interlocked in such a way that the programming model for them is identical for both integrated processors (Processor 0 and Processor 1). The two processor's view of the Door Bell Registers are illustrated in [Figure 118 on page 823](#).



13.2.1 Door Bell Register Operation

When configuring the Door Bell Register mechanism in the IMU for use, the following steps are followed:

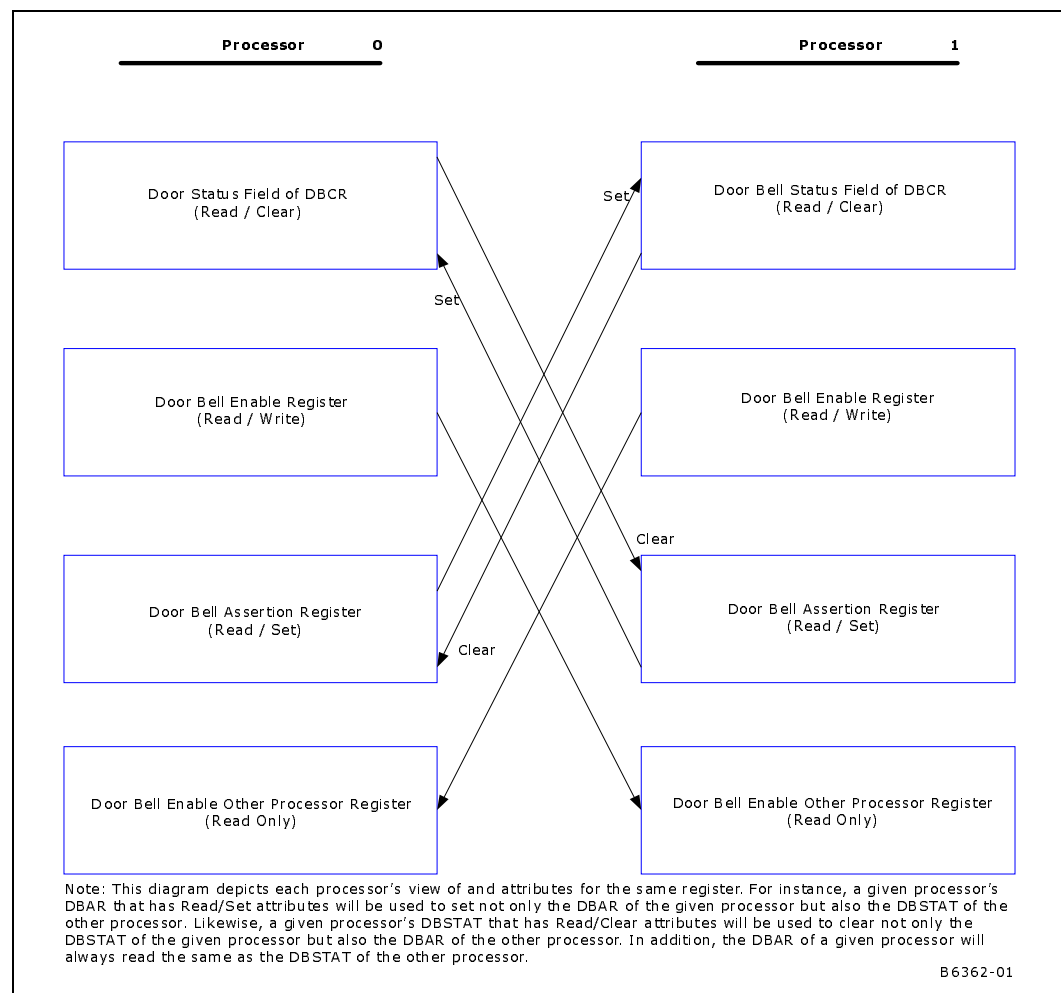
1. Set bits in DBER associated with the DBSTAT bits that will interrupt the processor.
2. Read DBEOR register to identify DBAR bits which will interrupt the other processor.

When interrupted by an enabled DBSTAT bit, the processor conducts activities associated with DBSTAT bit, and then clear the DBSTAT bit by writing a '1' to that bit.

When multiple DBSTAT bits are set, the processor may use the Highest Priority Doorbell bit field of the DBCR to handle the DBSTAT requests in a fixed priority order (see [Section 13.4.1, "Interrupt Prioritization" on page 829](#)).

The processor may set one or more DBSTAT bits in the other processor by writing to the DBAR register. The DBAR bits may only be cleared by the other processor.

Figure 118. Door Bell Registers as Viewed by Processor 0 and Processor 1





13.3 Circular Queues

Processor 0 and Processor 1 can send/receive messages from one another using the Circular Queue communication mechanism. There are a total of eight circular queues in the IMU. From the point of view of a given processor, there are four Send queues and four Receive Queues as detailed in [Table 504](#).

Table 504. Processor Send / Receive Queue Mapping

Processor 0	Processor 1
Send Queue 0	Receive Queue 0
Receive Queue 0	Send Queue 0
Send Queue 1	Receive Queue 1
Receive Queue 1	Send Queue 1
Send Queue 2	Receive Queue 2
Receive Queue 2	Send Queue 2
Send Queue 3	Receive Queue 3
Receive Queue 3	Send Queue 3

Note that in [Table 504](#) for a given processor's Send Queue, there is a corresponding Receive Queue in the other processor and visa versa.

The benefit of this mapping is that each processor views the eight queues (four Send Queues and four Receive Queues) in an identical manner.



13.3.1 Circular Queue Operation

A Circular Queue is a queue of messages. With the facilities in the IMU, the processor can derive message pointers to the queue of messages which reside in local memory.

As detailed in [Table 504](#), Processor 0 and Processor 1 each have four Receive Queues and four Send Queues. A given processor's Send queues are mapped to the other processor's Receive queues, and a given processor's Receive queues are mapped to the other processor's Send queues.

With this mapping, the IMU can provide an identical programming model for all eight queues to both processors. The registers provided for the management of a Send Queue are:

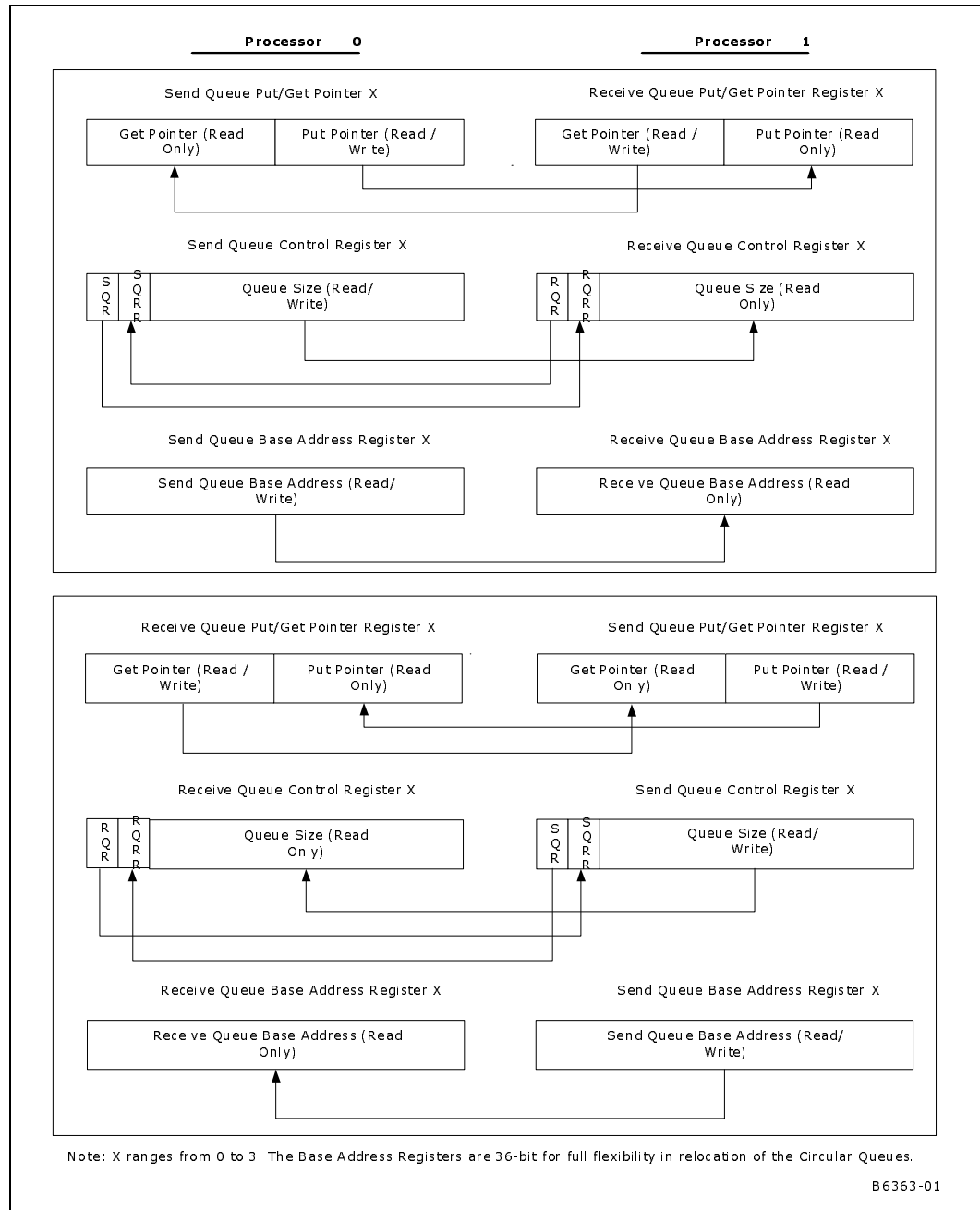
- **Send Queue Get/Put Pointer Register**
This register provides an interface for a processor to send message pointers to the other processor by reading the Get pointer and by writing the Put pointer. The Get and Put pointer values represent an Index that in conjunction with the Send Queue Upper/Lower Base Address Registers can be used to derive the message pointers. The Put and Get pointers are 16-bits wide which mean that each Circular queue may contain up to $(2^{16}-1)$ message pointers or 65535 message pointers.
- **Send Queue Control Register**
This register contains three fields, Size, Send Queue Reset Request, and a Send Queue Reset. The processor writes the Size field to set the last 16-bit Index in the Circular effectively fixing the size of the queue to the value of the Size field. The Send Queue Reset Request field is a request from the other processor to reinitialize the Put/Get Pointers for the Send Queue. Setting the Send Queue Reset field will reinitialize the Put/Get Pointers when the other processor is requesting that they be reinitialized.
- **Send Queue Upper/Lower Base Address Registers**
The processor writes these registers to set the location of the queue of messages for the Send Queue in the 36-bit local memory address space.

The registers provided for the management of a Receive Queue are:

- **Receive Queue Get/Put Pointer Register**
This register provides an interface for a processor to receive message pointers from the other processor by reading the Put pointer and by writing the Get pointer. The Get and Put pointer values represent an Index that in conjunction with the Receive Queue Upper/Lower Base Address Registers can be used to derive the message pointers. The Put and Get pointers are 16-bits wide which means that each Circular queue may contain up to $(2^{16}-1)$ message pointers or 65535 message pointers.
- **Receive Queue Control Register**
This register contains three fields, Size, Receive Queue Reset Request, and a Receive Queue Reset. The processor reads the Size field to obtain the last 16-bit Index in the Circular Queue. The Receive Queue Reset Request field is a request from the other processor to reinitialize the Put/Get Pointers for the Receive Queue. Setting the Receive Queue Reset field will reinitialize the Put/Get Pointers when the other processor is requesting that they be reinitialized.
- **Receive Queue Upper/Lower Base Address Registers**
The processor reads these registers to obtain the location of the queue of messages for the Receive Queue in the 36-bit local memory address space.

Figure 119 on page 826 illustrates the interlock of the Circular Queue register set between Processor 0 and Processor 1 for a single Send and Receive Queue pair

Figure 119. Circular Queue Control Registers as Viewed from Each Processor





13.3.1.1 Send Queue Management

Following a initialization, the Send Queue registers are set to their default value. The Send Queue will be configured and managed using the following steps:

1. Send Queue Upper/Lower Base Address Registers are written with the 36-bit local address (Send_Q_Base) where the Send Queue is located.
2. Send Queue Size field of the Send Queue Control Register is set to the last index in the queue of message pointers (Size).
3. A message is written to the local memory at the location pointed to by the message pointer derived from the Put pointer and the Send_Q_Base.²⁷
 - The Send Queue Not Full Interrupt may be used to inform the processor that there are available slots in the Send Queue.
4. The Put Pointer is incremented one or more times and written back to the Send Queue Get/Put Pointer register

Steps 3 and 4 may be repeated until the Send Queue is Full (Put Pointer — Get Pointer equal to Size). If necessary, the Send Queue may be reinitialized by writing to the Send Queue Reset bit in the Send Queue Control Register. The reinitialization won't take effect until both the Send Queue Reset bit and the Send Queue Reset Request bit (set by the other processor) are set. When reinitialized, the Get/Put Pointers, the Send Queue Reset bit, and the Send Queue Reset Request bit will be returned to their default values.

13.3.1.2 Receive Queue Management

Following a initialization, the Receive Queue registers are set to their default value. The Receive Queue will be configured and managed using the following steps:

1. Read the 36-bit local address (Receive_Q_Base) where the Receive Queue is located from the Receive Queue Upper/Lower Base Address Registers.
2. Read the last index into the queue of messages from the Receive Queue Size field (Size) of the Receive Queue Control Register.
3. A message is read from the local memory at the location pointed to by the message pointer derived from the Get Pointer and the Receive_Q_Base.²⁸
 - The Receive Queue Not Empty Interrupt may be used to inform the processor that there are messages requiring service in the Receive Queue.
4. The Get Pointer is incremented and written back to the Receive Queue Get/Put Pointer register.

Steps 3 and 4 may be repeated until the Receive Queue is empty (Put Pointer equal to Get Pointer). If necessary, the Receive Queue may be reinitialized by writing to the Receive Queue Reset bit in the Receive Queue Control Register. The reinitialization won't take effect until both the Receive Queue Reset bit and the Receive Queue Reset Request bit (set by the other processor) are set. When reinitialized, the Get/Put Pointers, the Receive Queue Reset bit, and the Receive Queue Reset Request bit will be returned to their default values.

27. The method by which the user combines the Put Pointer and the Send_Q_Base to derive the message pointer is not defined. The software needs to be aware of a message size.

28. The method by which the user combines the Get Pointer and the Receive_Q_Base to derive the message pointer is not defined. The software needs to be aware of a message size.



13.4 Interrupts

The IMU can generate two interrupts (one per Intel XScale® processor) for the assertion of Send Queue or Receive Queue Not Full or Not Empty conditions, respectively. In addition, the interrupt may be asserted when any of the 15 Door Bell Status bits are set by the other processor. The IMU has one interrupt output connected to the Interrupt Controller Unit for each core described in [Chapter 4, “Interrupt Controller Unit”](#). [Table 505](#) summarizes the status flags and conditions when interrupts are generated in the Door Bell Control Register (DBCR).

Table 505. IMU Interrupt Summary

Interrupt Condition	Door Bell Control Register Flags								
	Receive Queue 3 Not Empty	Send Queue 3 Not Full	Receive Queue 2 Not Empty	Send Queue 2 Not Full	Receive Queue 1 Not Empty	Send Queue 1 Not Full	Receive Queue 0 Not Empty	Send Queue 0 Not Full	Door Bell Status (14:0)
Receive Queue 3 Not Empty ^a (RQ3NE)	1	0	0	0	0	0	0	0	0
Send Queue 3 Not Full ^b (SQ3NF)	0	1	0	0	0	0	0	0	0
Receive Queue 2 Not Empty (RQ2NE)	0	0	1	0	0	0	0	0	0
Send Queue 2 Not Full (SQ2NF)	0	0	0	1	0	0	0	0	0
Receive Queue 1 Not Empty (RQ1NE)	0	0	0	0	1	0	0	0	0
Send Queue 1 Not Full (SQ1NF)	0	0	0	0	0	1	0	0	0
Receive Queue 0 Not Empty (RQ0NE)	0	0	0	0	0	0	1	0	0
Send Queue 0 Not Full (SQ0NF)	0	0	0	0	0	0	0	1	0
Door Bell Status! = 0	0	0	0	0	0	0	0	0	See Note ^c

a. Empty Condition is (Put Pointer = Get Pointer)

b. Full Condition is (Put Pointer — Get Pointer = Size)

c. Appropriate bits in Door Bell Status (14:0) are set depending on which door bells the other processor set in its' Door Bell Assert Register.



13.4.1 Interrupt Prioritization

The IMU provides a fixed priority scheme for servicing the 23 Interrupt Status bits in the DBCR

For the eight Circular Queue Attention bits (DBCR[23:16]), the 4-bit Highest Priority Circular Queue Attention field (DBCR[27:24]) represents the lowest order bit set.

For the Doorbell Status field of the DBCR[14:0], the 4-bit Highest Priority Door Bell field (DBCR[31:28]) represents the lowest order bit set.

The two fixed priority schemes are detailed in [Table 506](#).

Table 506. Fixed Priority Scheme for Door Bell Status bits and Circular Queue Attention Bits

Priority	Door Bell Status Bit Set	Highest Priority Door Bell Field	Circular Attention Bit Set	Highest Priority Circular Queue Attention Field
Highest	DBSTAT0	0h	SQ0NF	0h
	DBSTAT1	1h	RQ0NE	1h
	DBSTAT2	2h	SQ1NF	2h
	DBSTAT3	3h	RQ1NE	3h
	DBSTAT4	4h	SQ2NF	4h
	DBSTAT5	5h	RQ2NE	5h
	DBSTAT6	6h	SQ3NF	6h
	DBSTAT7	7h	RQ3NE	7h
	DBSTAT8	8h	-	-
	DBSTAT9	9h	-	-
	DBSTAT10	Ah	-	-
	DBSTAT11	Bh	-	-
	DBSTAT12	Ch	-	-
	DBSTAT13	Dh	-	-
Lowest	DBSTAT14	Eh	-	-
-	None	Fh	None	Fh



13.5 Power-up/Default Status

Upon power-up, an external hardware reset, the IMU Registers will be initialized to their default values.

13.6 Register Definitions

The IMU contains registers for sending and receiving doorbells, managing the circular queues and controlling the Intel XScale® processor interrupts.

The location of these registers are specified as a relative offset to a 512 KB aligned global PMMR offset. The default for the 512 KB aligned offset is 0 FFD8 0000H defined by the PMMRBAR register. See also [Chapter 21.0, "Peripheral Registers"](#).

Note: The IMU Registers can only be accessed by the Intel XScale® processors. For example, other initiators cannot access the IMU registers as the IMU will terminate the request with an address error.

Table 507. Inter-processor Messaging Unit Registers (Sheet 1 of 2)

Section, Register Name, Acronym, Page
Section 13.6.1, "Door Bell Control Register — DBCR" on page 832
Section 13.6.2, "Door Bell Enable Register — DBER" on page 834
Section 13.6.3, "Door Bell Assertion Register — DBAR" on page 835
Section 13.6.4, "Door Bell Enable Other Processor Register — DBEOR" on page 835
Section 13.6.5, "Send Queue Put/Get Pointer Register 0 — SQPG0" on page 836
Section 13.6.6, "Send Queue Control Register 0 — SQCR0" on page 837
Section 13.6.7, "Send Queue Lower Base Address Register 0 — SQLBAR0" on page 838
Section 13.6.8, "Send Queue Upper Base Address Register 0 — SQUBAR0" on page 838
Section 13.6.9, "Receive Queue Put/Get Pointer Register 0 — RQPG0" on page 839
Section 13.6.10, "Receive Queue Control Register 0 — RQCR0" on page 840
Section 13.6.11, "Receive Queue Lower Base Address Register 0 — RQLBAR0" on page 840
Section 13.6.12, "Receive Queue Upper Base Address Register 0 — RQUBAR0" on page 841
Section 13.6.13, "Send Queue Put/Get Pointer Register 1 — SQPG1" on page 841
Section 13.6.14, "Send Queue Control Register 1 — SQCR1" on page 842
Section 13.6.15, "Send Queue Lower Base Address Register 1 — SQLBAR1" on page 843
Section 13.6.16, "Send Queue Upper Base Address Register 1 — SQUBAR1" on page 843
Section 13.6.17, "Receive Queue Put/Get Pointer Register 1 — RQPG1" on page 844
Section 13.6.18, "Receive Queue Control Register 1 — RQCR1" on page 845
Section 13.6.19, "Receive Queue Lower Base Address Register 1 — RQLBAR1" on page 846
Section 13.6.20, "Receive Queue Upper Base Address Register 1 — RQUBAR1" on page 846
Section 13.6.21, "Send Queue Put/Get Pointer Register 2 — SQPG2" on page 847
Section 13.6.22, "Send Queue Control Register 2 — SQCR2" on page 848
Section 13.6.23, "Send Queue Lower Base Address Register 2 — SQLBAR2" on page 849
Section 13.6.24, "Send Queue Upper Base Address Register 2 — SQUBAR2" on page 849
Section 13.6.25, "Receive Queue Put/Get Pointer Register 2 — RQPG2" on page 850
Section 13.6.26, "Receive Queue Control Register 2 — RQCR2" on page 851
Section 13.6.27, "Receive Queue Lower Base Address Register 2 — RQLBAR2" on page 852

**Table 507. Inter-processor Messaging Unit Registers (Sheet 2 of 2)**

Section, Register Name, Acronym, Page
Section 13.6.28, "Receive Queue Upper Base Address Register 2 — RQUBAR2" on page 852
Section 13.6.29, "Send Queue Put/Get Pointer Register 3 — SQPG3" on page 853
Section 13.6.30, "Send Queue Control Register 3 — SQCR3" on page 854
Section 13.6.31, "Send Queue Lower Base Address Register 3 — SQLBAR3" on page 854
Section 13.6.32, "Send Queue Upper Base Address Register 3 — SQUBAR3" on page 855
Section 13.6.33, "Receive Queue Put/Get Pointer Register 3 — RQPG3" on page 855
Section 13.6.34, "Receive Queue Control Register 3 — RQCR3" on page 856
Section 13.6.35, "Receive Queue Lower Base Address Register 3 — RQLBAR3" on page 856
Section 13.6.36, "Receive Queue Upper Base Address Register 3 — RQUBAR3" on page 857
Section 13.6.37, "IMU Test and Set Registers — IMUTSR[0:511]" on page 858



13.6.1 Door Bell Control Register — DBCR

The Door Bell Control Register (DBCR) contains the Door Bell and Circular Queue interrupt status bits that are used to manage the two messaging schemes. The DBCR includes the output of a fixed prioritization of the Doorbell and Circular Queue interrupt status bits, respectively. In addition, Door Bell interrupt status bits may be cleared via the DBCR.

Table 508. Door Bell Control Register — DBCR (Sheet 1 of 2)

		Intel XScale® processor internal bus address offset +0A00H	
Bit	Default	Description	
31:28	1111 ₂	Highest Priority Door Bell -- Performs a fixed priority decode of the 15-bit Door Bell Status field (DBSTAT) of the DBCR. Highest priority is the lowest order Door Bell Status Bit Set.	
		0	DBSTAT0
		1	DBSTAT1
		2	DBSTAT2
		3	DBSTAT3
		4	DBSTAT4
		5	DBSTAT5
		6	DBSTAT6
		7	DBSTAT7
		8	DBSTAT8
		9	DBSTAT9
		10	DBSTAT10
		11	DBSTAT11
		12	DBSTAT12
		13	DBSTAT13
		14	DBSTAT14
		15	None (Default Value)
27:24	1111 ₂	Highest Priority Circular Queue Attention -- Performs a fixed priority decode of the 8 Circular Queue Attention Bits (bits 23 to 16 of the DBCR). Highest priority is the lowest order Bit Set.	
		0	S0NF
		1	R0NE
		2	S1NF
		3	R1NE
		4	S2NF
		5	R2NE
		6	S3NF
23:16		7	R3NE
		8	Reserved
		9	Reserved
		10	Reserved
		11	Reserved
		12	Reserved
		13	Reserved
		14	Reserved
		15	None (Default Value)
23	0 ₂	Receive Queue 3 Not Empty (R3NE) -- When set, Receive Queue 3 is not empty (Receive Queue 3 Get Pointer Does not equal Receive Queue 3 Put Pointer)	
22	0 ₂	Send Queue 3 Not Full (S3NF) -- When set, Send Queue 3 is not Full and ES3NF is set in the Door Bell Enable Register. (Send Queue 3 Put Pointer minus Send Queue 3 Get Pointer does not equal Send Queue 3 Size)	
21	0 ₂	Receive Queue 2 Not Empty (R2NE) -- When set, Receive Queue 2 is not empty (Receive Queue 2 Get Pointer Does not equal Receive Queue 2 Put Pointer)	
20	0 ₂	Send Queue 2 Not Full (S2NF) -- When set, Send Queue 2 is not Full and ES2NF is set in the Door Bell Enable Register. (Send Queue 2 Put Pointer minus Send Queue 2 Get Pointer does not equal Send Queue 2 Size)	
19	0 ₂	Receive Queue 1 Not Empty (R1NE) -- When set, Receive Queue 1 is not empty (Receive Queue 1 Get Pointer Does not equal Receive Queue 1 Put Pointer)	

**Table 508. Door Bell Control Register — DBCR (Sheet 2 of 2)**

Intel XScale® processor internal bus address offset +0A00H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
18	0 ₂	Send Queue 1 Not Full (S1NF) -- When set, Send Queue 1 is not Full and ES1NF is set in the Door Bell Enable Register. (Send Queue 1 Put Pointer minus Send Queue 1 Get Pointer does not equal Send Queue 1 Size)
17	0 ₂	Receive Queue 0 Not Empty (R0NE) -- When set, Receive Queue 0 is not empty (Receive Queue 0 Get Pointer Does not equal Receive Queue 0 Put Pointer)
16	0 ₂	Send Queue 0 Not Full (S0NF) -- When set, Send Queue 0 is not Full and ES0NF is set in the Door Bell Enable Register. (Send Queue 0 Put Pointer minus Send Queue 0 Get Pointer does not equal Send Queue 0 Size)
15	0 ₂	Reserved
14:00	0000H	Door Bell Status (DBSTAT) -- These bits show the Door Bell Status set by the other processor. A Door Bell status bit may be cleared by writing a '1' to that bit. Note: Doorbell status bits can be set only by the other processor



13.6.2 Door Bell Enable Register — DBER

The Door Bell Enable Register (DBER) contains the Door Bell and Circular Queue interrupt enable bits that are used to manage the two messaging schemes. The Circular Queue Not Full Interrupt Enables are also used as a condition of setting their associated Circular Not Full Interrupt Status bits in the DBCR.

Table 509. Door Bell Enable Register — DBER

IOP Attributes PCI Attributes 31 28 24 20 16 12 8 4 0 rv rv rv rv rv rv rv rv rw rw rw rw rw rw rw rv rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor internal bus address offset +0A04H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:24	00H	Reserved
23	0 ₂	Enable Receive Queue 3 Not Empty Interrupt (ER3NE) -- When set, an interrupt request is asserted to the Intel XScale® processor when the Receive Queue 3 Not Empty Status bit is set in the DBCR.
22	0 ₂	Enable Send Queue 3 Not Full Interrupt (ES3NF) -- When set, the Send Queue 3 Not Full Status bit is set in the DBCR and an interrupt request is asserted to the Intel XScale® processor when Send Queue 3 is not Full.
21	0 ₂	Enable Receive Queue 2 Not Empty Interrupt (ER2NE) -- When set, an interrupt request is asserted to the Intel XScale® processor when the Receive Queue 2 Not Empty Status bit is set in the DBCR.
20	0 ₂	Enable Send Queue 2 Not Full Interrupt (ES2NF) -- When set, the Send Queue 2 Not Full Status bit is set in the DBCR and an interrupt request is asserted to the Intel XScale® processor when Send Queue 2 is not Full.
19	0 ₂	Enable Receive Queue 1 Not Empty Interrupt (ER1NE) -- When set, an interrupt request is asserted to the Intel XScale® processor when the Receive Queue 1 Not Empty Status bit is set in the DBCR.
18	0 ₂	Enable Send Queue 1 Not Full Interrupt (ES1NF) -- When set, the Send Queue 1 Not Full Status bit is set in the DBCR and an interrupt request is asserted to the Intel XScale® processor when Send Queue 1 is not Full.
17	0 ₂	Enable Receive Queue 0 Not Empty Interrupt (ER0NE) -- When set, an interrupt request is asserted to the Intel XScale® processor when the Receive Queue 0 Not Empty Status bit is set in the DBCR.
16	0 ₂	Enable Send Queue 0 Not Full Interrupt (ES0NF) -- When set, the Send Queue 0 Not Full Status bit is set in the DBCR and an interrupt request is asserted to the Intel XScale® processor when Send Queue 0 is not Full.
15	0 ₂	Reserved
14:00	0000H	Enable Door Bell Status Interrupt (EDBST) -- When any of these bits is set, an interrupt is asserted to the Intel XScale® processor when the corresponding bit in the Door Bell Status (DBST) field of the DBCR is set.



13.6.5 Send Queue Put/Get Pointer Register 0 — SQPG0

The Send Queue Put/Get Pointer Register 0 (SQPG0) provides the capability to communicate to the other processor through the Put pointer that there have been one or more entries added to Send Queue 0. Likewise, the other processor can communicate back through the Get pointer which Send Queue 0 entries are now released for adding new messages.

Table 512. Send Queue Put/Get Pointer Register 0 — SQPG0

312824201612840 IOP Attributes ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor internal bus address offset +0A20H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																													
31:16	0000H	Send Queue 0 Get Pointer — Index of the next queue entry for the other processor to read in Send Queue 0.																													
15:00	0000H	Send Queue 0 Put Pointer — Index of the next queue entry to fill in Send Queue 0.																													



13.6.6 Send Queue Control Register 0 — SQCR0

The Send Queue Control Register 0 (SQCR0) provides the ability to reset the Send Queue 0 Put/Get pointers at the request of the other processor. In addition, the size of Send Queue 0 is configured by the Send Queue 0 Size field of the SQCR0.

Table 513. Send Queue Control Register 0 — SQCR0

Intel XScale® processor internal bus address offset +0A24H		
Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Send Queue 0 Reset (SQ0R) — Reinitialize Send Queue 0 by returning the Put/Get pointers to their default values. Note: The reinitialization of Send Queue 0 will not take effect until the Send Queue 0 Reset Request bit (SQ0RR) in the SQCR0 is set by the other processor. The SQ0R and SQ0RR bits in the SQCR0 will be reinitialized along with the Put/Get Pointers.
30	0 ₂	Send Queue 0 Reset Request (SQ0RR) — The other processor is requesting that Send Queue 0 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Send Queue 0 will not take effect until the Send Queue 0 Reset bit (SQ0R) in the SQCR0 is set. The SQ0R and SQ0RR bits in the SQCR0 will be reinitialized along with the Put/Get Pointers.
29:15	0000H	Reserved
15:00	0000H	Send Queue 0 Size — Index of the last queue entry in Send Queue 0.



13.6.7 Send Queue Lower Base Address Register 0 — SQLBAR0

The Send Queue Lower Base Address Register 0 (SQLBAR0) sets the lower 32-bits of the address for the first queue entry in Send Queue 0.

Table 514. Send Queue Lower Base Address Register 0 — SQLBAR0

Bit	Default	Description																													
31:00	00000000H	Send Queue 0 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Send Queue 0.																													

13.6.8 Send Queue Upper Base Address Register 0 — SQUBAR0

The Send Queue Upper Base Address Register 0 (SQUBAR0) sets the upper 4-bits of the address for the first queue entry in Send Queue 0.

Table 515. Send Queue Upper Base Address Register 0 — SQUBAR0

Bit	Default	Description																													
31:4	00000000H	Reserved																													
3:0	0H	Send Queue 0 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Send Queue 0.																													



13.6.9 Receive Queue Put/Get Pointer Register 0 – RQPG0

The Receive Queue Put/Get Pointer Register 0 (RQPG0) provides the capability to communicate to the other processor through the Get pointer that one or more active entries of Receive Queue 0 have been processed and are now available to the Put pointer. Likewise, the other processor can communicate through the Put pointer that one or more additional messages have been added to Receive Queue 0.

Table 516. Receive Queue Put/Get Pointer Register 0 – RQPG0

Intel XScale® processor internal bus address offset +0A30H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:16	0000H	Receive Queue 0 Get Pointer — Index of the next queue entry to read in Receive Queue 0.
15:00	0000H	Receive Queue 0 Put Pointer — Index of the next queue entry for the other processor to fill in Receive Queue 0.



13.6.10 Receive Queue Control Register 0 — RQCR0

The Receive Queue Control Register 0 (RQCR0) provides the ability to reset the Receive Queue 0 Put/Get pointers at the request of the other processor. In addition, the size of Receive Queue 0 is represented by the Receive Queue 0 Size field of the RQCR0.

Table 517. Receive Queue Put/Get Pointer Register 0 — RQPG0

Intel XScale® processor internal bus address offset +0A34H																Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set								RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible							
Bit	Default	Description																													
31	0 ₂	Receive Queue 0 Reset (RQ0R) — Reinitialize Receive Queue 0 by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 0 will not take effect until the Receive Queue 0 Reset Request bit (RQ0RR) in the RQCR0 is set by the other processor. The RQ0R and RQ0RR bits in the RQCR0 will be reinitialized along with the Put/Get Pointers.																													
30	0 ₂	Receive Queue 0 Reset Request (RQ0RR) — The other processor is requesting that Receive Queue 0 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 0 will not take effect until the Receive Queue 0 Reset bit (RQ0R) in the RQCR0 is set. The RQ0R and RQ0RR bits in the RQCR0 will be reinitialized along with the Put/Get Pointers.																													
29:15	0000H	Reserved																													
15:00	0000H	Receive Queue 0 Size — Index of the last queue entry in Receive Queue 0.																													

13.6.11 Receive Queue Lower Base Address Register 0 — RQLBAR0

The Receive Queue Lower Base Address Register 0 (RQLBAR0) represents the lower 32-bits of the address for the first queue entry in Receive Queue 0.

Table 518. Receive Queue Lower Base Address Register 0 — RQLBAR0

		IOP Attributes PCI Attributes 312824201612840 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na na Intel XScale® processor internal bus address offset +0A38H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible																															
Bit	Default	Description																															
31:00	00000000H	Receive Queue 0 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Receive Queue 0.																															



13.6.12 Receive Queue Upper Base Address Register 0 — RQUBAR0

The Receive Queue Upper Base Address Register 0 (RQUBAR0) represents the upper 4-bits of the address for the first queue entry in Receive Queue 0.

Table 519. Receive Queue Upper Base Address Register 0 — RQUBAR0

		Intel XScale® processor internal bus address offset +0A3CH	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description	
31:4	00000000H	Reserved	
3:0	0H	Receive Queue 0 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Receive Queue 0.	

13.6.13 Send Queue Put/Get Pointer Register 1 — SQPG1

The Send Queue Put/Get Pointer Register 1 (SQPG1) provides the capability to communicate to the other processor through the Put pointer that there have been one or more entries added to Send Queue 1. Likewise, the other processor can communicate back through the Get pointer which Send Queue 1 entries are now released for adding new messages.

Table 520. Send Queue Put/Get Pointer Register 1 — SQPG1

		Intel XScale® processor internal bus address offset +0A40H	Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description	
31:16	0000H	Send Queue 1 Get Pointer — Index of the next queue entry for the other processor to read in Send Queue 1.	
15:00	0000H	Send Queue 1 Put Pointer — Index of the next queue entry to fill in Send Queue 1.	



The Send Queue Control Register 1 (SQCR1) provides the ability to reset the Send Queue 1 Put/Get pointers at the request of the other processor. In addition, the size of Send Queue 1 is configured by the Send Queue 1 Size field of the SQCR1.

Intel XScale[®] processor internal bus address offset
+0A44H

Attribute Legend:
RW = Read/Write
RC = Read Clear
RO = Read Only
RS = Read/Set
RV = Reserved
PR = Preserved
NA = Not Accessible



13.6.15 Send Queue Lower Base Address Register 1 — SQLBAR1

The Send Queue Lower Base Address Register 1 (SQLBAR1) sets the lower 32-bits of the address for the first queue entry in Send Queue 1.

Table 522. Send Queue Lower Base Address Register 1 — SQLBAR1

		Intel XScale® processor internal bus address offset +0A48H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:00	00000000H	Send Queue 1 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Send Queue 1.

13.6.16 Send Queue Upper Base Address Register 1 — SQUBAR1

The Send Queue Upper Base Address Register 1 (SQUBAR1) sets the upper 4-bits of the address for the first queue entry in Send Queue 1.

Table 523. Send Queue Upper Base Address Register 1 — SQUBAR1

		Intel XScale® processor internal bus address offset +0A4CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:4	00000000H	Reserved
3:0	0H	Send Queue 1 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Send Queue 1.



13.6.17 Receive Queue Put/Get Pointer Register 1 — RQPG1

The Receive Queue Put/Get Pointer Register 1 (RQPG1) provides the capability to communicate to the other processor through the Get pointer that one or more active entries of Receive Queue 1 have been processed and are now available to the Put pointer. Likewise, the other processor can communicate through the Put pointer that one or more additional messages have been added to Receive Queue 1.

Table 524. Receive Queue Put/Get Pointer Register 1 — RQPG1

		Intel XScale® processor internal bus address offset +0A50H																											
Bit	Default	Description																											
31:16	0000H	Receive Queue 1 Get Pointer — Index of the next queue entry to read in Receive Queue 1.																											
15:00	0000H	Receive Queue 1 Put Pointer — Index of the next queue entry for the other processor to fill in Receive Queue 1.																											



13.6.18 Receive Queue Control Register 1 — RQCR1

The Receive Queue Control Register 1 (RQCR1) provides the ability to reset the Receive Queue 1 Put/Get pointers at the request of the other processor. In addition, the size of Receive Queue 1 is represented by the Receive Queue 1 Size field of the RQCR1.

Table 525. Receive Queue Put/Get Pointer Register 1 — RQPG1

Intel XScale[®] processor internal bus address offset +0A54H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Receive Queue 1 Reset (RQ1R) — Reinitialize Receive Queue 1 by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 1 will not take effect until the Receive Queue 1 Reset Request bit (RQ1RR) in the RQCR1 is set by the other processor. The RQ1R and RQ1RR bits in the RQCR1 will be reinitialized along with the Put/Get Pointers.
30	0 ₂	Receive Queue 1 Reset Request (RQ1RR) — The other processor is requesting that Receive Queue 1 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 1 will not take effect until the Receive Queue 1 Reset bit (RQ1R) in the RQCR1 is set. The RQ1R and RQ1RR bits in the RQCR1 will be reinitialized along with the Put/Get Pointers.
29:15	0000H	Reserved
15:00	0000H	Receive Queue 1 Size — Index of the last queue entry in Receive Queue 1.



13.6.19 Receive Queue Lower Base Address Register 1 — RQLBAR1

The Receive Queue Lower Base Address Register 1 (RQLBAR1) represents the lower 32-bits of the address for the first queue entry in Receive Queue 1.

Table 526. Receive Queue Lower Base Address Register 1 — RQLBAR1

Bit	Default	Description
31:00	00000000H	Receive Queue 1 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Receive Queue 1.

13.6.20 Receive Queue Upper Base Address Register 1 — RQUBAR1

The Receive Queue Upper Base Address Register 1 (RQUBAR1) represents the upper 4-bits of the address for the first queue entry in Receive Queue 1.

Table 527. Receive Queue Upper Base Address Register 1 — RQUBAR1

Bit	Default	Description
31:4	00000000H	Reserved
3:0	0H	Receive Queue 1 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Receive Queue 1.



13.6.21 Send Queue Put/Get Pointer Register 2 — SQPG2

The Send Queue Put/Get Pointer Register 2 (SQPG2) provides the capability to communicate to the other processor through the Put pointer that there have been one or more entries added to Send Queue 2. Likewise, the other processor can communicate back through the Get pointer which Send Queue 2 entries are now released for adding new messages.

Table 528. Send Queue Put/Get Pointer Register 2 — SQPG2

Bit	Default	Description																											
31:16	0000H	Send Queue 2 Get Pointer — Index of the next queue entry for the other processor to read in Send Queue 2.																											
15:00	0000H	Send Queue 2 Put Pointer — Index of the next queue entry to fill in Send Queue 2.																											



13.6.22 Send Queue Control Register 2 — SQCR2

The Send Queue Control Register 2 (SQCR2) provides the ability to reset the Send Queue 2 Put/Get pointers at the request of the other processor. In addition, the size of Send Queue 2 is configured by the Send Queue 2 Size field of the SQCR2.

Table 529. Send Queue Control Register 2 — SQCR2

Intel XScale® processor internal bus address offset +0A64H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Send Queue 2 Reset (SQ2R) — Reinitialize Send Queue 2 by returning the Put/Get pointers to their default values. Note: The reinitialization of Send Queue 2 will not take effect until the Send Queue 2 Reset Request bit (SQ2RR) in the SQCR2 is set by the other processor. The SQ2R and SQ2RR bits in the SQCR2 will be reinitialized along with the Put/Get Pointers.
30	0 ₂	Send Queue 2 Reset Request (SQ2RR) — The other processor is requesting that Send Queue 2 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Send Queue 2 will not take effect until the Send Queue 2 Reset bit (SQ2R) in the SQCR2 is set. The SQ2R and SQ2RR bits in the SQCR2 will be reinitialized along with the Put/Get Pointers.
29:15	0000H	Reserved
15:00	0000H	Send Queue 2 Size — Index of the last queue entry in Send Queue 2.

13.6.23 Send Queue Lower Base Address Register 2 — SQLBAR2

The Send Queue Lower Base Address Register 2 (SQLBAR2) sets the lower 32-bits of the address for the first queue entry in Send Queue 2.

Table 530. Send Queue Lower Base Address Register 2 – SQLBAR2

Intel XScale® processor internal bus address offset +0A68H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	Send Queue 2 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Send Queue 2.

13.6.24 Send Queue Upper Base Address Register 2 — SQUBAR2

The Send Queue Upper Base Address Register 2 (SQUBAR2) sets the upper 4-bits of the address for the first queue entry in Send Queue 2.

Table 531. Send Queue Upper Base Address Register 2 — SQUBAR2

Intel XScale® processor internal bus address offset
+0A6CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible



13.6.25 Receive Queue Put/Get Pointer Register 2 — RQPG2

The Receive Queue Put/Get Pointer Register 2 (RQPG2) provides the capability to communicate to the other processor through the Get pointer that one or more active entries of Receive Queue 2 have been processed and are now available to the Put pointer. Likewise, the other processor can communicate through the Put pointer that one or more additional messages have been added to Receive Queue 2.

Table 532. Receive Queue Put/Get Pointer Register 2 — RQPG2

Bit	Default	Description																													
31:16	0000H	Receive Queue 2 Get Pointer — Index of the next queue entry to read in Receive Queue 2.																													
15:00	0000H	Receive Queue 2 Put Pointer — Index of the next queue entry for the other processor to fill in Receive Queue 2.																													



13.6.26 Receive Queue Control Register 2 — RQCR2

The Receive Queue Control Register 2 (RQCR2) provides the ability to reset the Receive Queue 2 Put/Get pointers at the request of the other processor. In addition, the size of Receive Queue 2 is represented by the Receive Queue 2 Size field of the RQCR2.

Table 533. Receive Queue Put/Get Pointer Register 2 — RQPG2

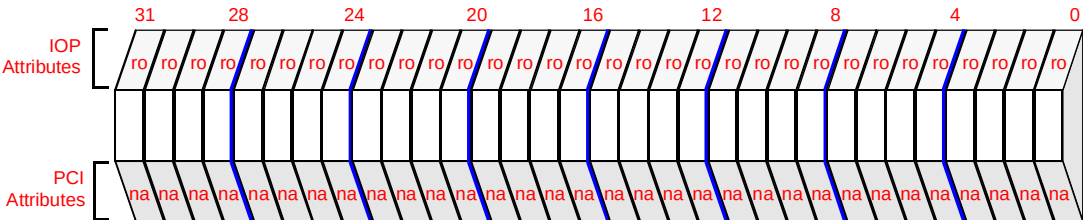
Intel XScale® processor internal bus address offset +0A74H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Receive Queue 2 Reset (RQ2R) — Reinitialize Receive Queue 2 by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 2 will not take effect until the Receive Queue 2 Reset Request bit (RQ2RR) in the RQCR2 is set by the other processor. The RQ2R and RQ2RR bits in the RQCR2 will be reinitialized along with the Put/Get Pointers.
30	0 ₂	Receive Queue 2 Reset Request (RQ2RR) — The other processor is requesting that Receive Queue 2 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 2 will not take effect until the Receive Queue 2 Reset bit (RQ2R) in the RQCR2 is set. The RQ2R and RQ2RR bits in the RQCR2 will be reinitialized along with the Put/Get Pointers.
29:15	0000H	Reserved
15:00	0000H	Receive Queue 2 Size — Index of the last queue entry in Receive Queue 2.



13.6.27 Receive Queue Lower Base Address Register 2 — RQLBAR2

The Receive Queue Lower Base Address Register 2 (RQLBAR2) represents the lower 32-bits of the address for the first queue entry in Receive Queue 2.

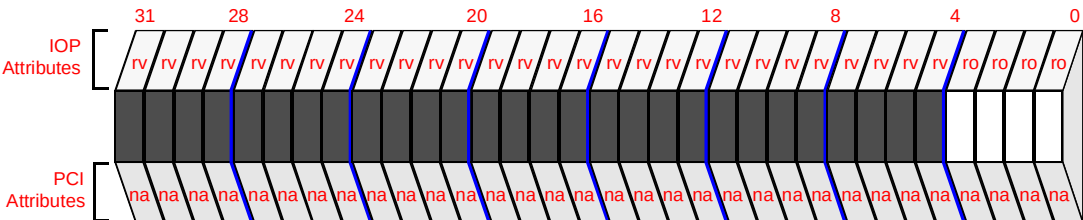
Table 534. Receive Queue Lower Base Address Register 2 — RQLBAR2

 Intel XScale® processor internal bus address offset +0A78H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	Receive Queue 2 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Receive Queue 2.

13.6.28 Receive Queue Upper Base Address Register 2 — RQUBAR2

The Receive Queue Upper Base Address Register 2 (RQUBAR2) represents the upper 4-bits of the address for the first queue entry in Receive Queue 2.

Table 535. Receive Queue Upper Base Address Register 2 — RQUBAR2

 Intel XScale® processor internal bus address offset +0A7CH Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:4	00000000H	Reserved
3:0	0H	Receive Queue 2 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Receive Queue 2.



13.6.29 Send Queue Put/Get Pointer Register 3 — SQPG3

The Send Queue Put/Get Pointer Register 3 (SQPG3) provides the capability to communicate to the other processor through the Put pointer that there have been one or more entries added to Send Queue 3. Likewise, the other processor can communicate back through the Get pointer which Send Queue 3 entries are now released for adding new messages.

Table 536. Send Queue Put/Get Pointer Register 3 — SQPG3

Intel XScale® processor internal bus address offset
+0A80H

Attribute Legend:
 RW = Read/Write
 RV = Reserved
 PR = Preserved
 RS = Read/Set
 RC = Read Clear
 RO = Read Only
 NA = Not Accessible



13.6.30 Send Queue Control Register 3 — SQCR3

The Send Queue Control Register 3 (SQCR3) provides the ability to reset the Send Queue 3 Put/Get pointers at the request of the other processor. In addition, the size of Send Queue 3 is configured by the Send Queue 3 Size field of the SQCR3.

Table 537. Send Queue Control Register 3 — SQCR3

Intel XScale® processor internal bus address offset +0A84H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31	0 ₂	Send Queue 3 Reset (SQ3R): Returns Put/Get pointers to default values to reinitialize Send Queue 3. Note: The reinitialization of Send Queue 3 will not take effect until the Send Queue 3 Reset Request bit (SQ3RR) in the SQCR3 is set by the other processor. The SQ3R and SQ3RR bits in the SQCR3 will be reinitialized along with the Put/Get Pointers.
30	0 ₂	Send Queue 3 Reset Request (SQ3RR) — The other processor is requesting that Send Queue 3 be reinitialized by returning the Put/Get pointers to their default values. Note: Reinitialization of Send Queue 3 is not effective until Send Queue 3 Reset bit (SQ3R) in SQCR3 is set. SQ3R and SQ3RR bits in SQCR3 are reinitialized along with Put/Get Pointers.
29:15	0000H	Reserved
15:00	0000H	Send Queue 3 Size — Index of the last queue entry in Send Queue 3.

13.6.31 Send Queue Lower Base Address Register 3 — SQLBAR3

The Send Queue Lower Base Address Register 3 (SQLBAR3) sets the lower 32-bits of the address for the first queue entry in Send Queue 3.

Table 538. Send Queue Lower Base Address Register 3 — SQLBAR3

Intel XScale® processor internal bus address offset +0A88H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:00	00000000H	Send Queue 3 Base Lower Base Address: Send Queue 3 lower 32-bits of first queue entry address.



13.6.32 Send Queue Upper Base Address Register 3 — SQUBAR3

The Send Queue Upper Base Address Register 3 (SQUBAR3) sets the upper 4-bits of the address for the first queue entry in Send Queue 3.

Table 539. Send Queue Upper Base Address Register 3 — SQUBAR3

Bit	Default	Description
31:4	00000000H	Reserved
3:0	0H	Send Queue 3 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Send Queue 3.

13.6.33 Receive Queue Put/Get Pointer Register 3 — RQPG3

The Receive Queue Put/Get Pointer Register 3 (RQPG3) provides the capability to communicate to the other processor through the Get pointer that one or more active entries of Receive Queue 3 have been processed and are now available to the Put pointer. Likewise, the other processor can communicate through the Put pointer that one or more additional messages have been added to Receive Queue 3.

Table 540. Receive Queue Put/Get Pointer Register 3 — RQPG3

Bit	Default	Description
31:16	0000H	Receive Queue 3 Get Pointer — Index of next queue entry to read in Receive Queue 3.
15:3	0000H	Receive Queue 3 Put Pointer — Index of next queue entry for the other processor to fill in Receive Queue 3.



13.6.34 Receive Queue Control Register 3 — RQCR3

The Receive Queue Control Register 3 (RQCR3) provides the ability to reset the Receive Queue 3 Put/Get pointers at the request of the other processor. In addition, the size of Receive Queue 3 is represented by the Receive Queue 3 Size field of the RQCR3.

Table 541. Receive Queue Put/Get Pointer Register 3 — RQPG3

IOP Attributes 312824201612840 rs ro rv rv rv rv rv rv rv rv rv rv rv rv rv ro ro ro ro ro ro ro ro ro ro ro ro PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na		Intel XScale® processor internal bus address offset +0A94H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set		RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description					
31	0 ₂	Receive Queue 3 Reset (RQ3R) — Reinitialize Receive Queue 3 by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 3 will not take effect until the Receive Queue 3 Reset Request bit (RQ3RR) in the RQCR3 is set by the other processor. The RQ3R and RQ3RR bits in the RQCR3 will be reinitialized along with the Put/Get Pointers.					
30	0 ₂	Receive Queue 3 Reset Request (RQ3RR) — The other processor is requesting that Receive Queue 3 be reinitialized by returning the Put/Get pointers to their default values. Note: The reinitialization of Receive Queue 3 will not take effect until the Receive Queue 3 Reset bit (RQ3R) in the RQCR3 is set. The RQ3R and RQ3RR bits in the RQCR3 will be reinitialized along with the Put/Get Pointers.					
29:15	0000H	Reserved					
15:00	0000H	Receive Queue 3 Size — Index of the last queue entry in Receive Queue 3.					

13.6.35 Receive Queue Lower Base Address Register 3 — RQLBAR3

The Receive Queue Lower Base Address Register 3 (RQLBAR3) represents the lower 32-bits of the address for the first queue entry in Receive Queue 3.

Table 542. Receive Queue Lower Base Address Register 3 — RQLBAR3

IOP Attributes 312824201612840 ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro ro PCI Attributes na na na na na na na na na na na na na na na na na na na na na na na na na na na na		Intel XScale® processor internal bus address offset +0A98H Attribute Legend: RW = Read/Write RV = Reserved PR = Preserved RS = Read/Set RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:00	00000000H	Receive Queue 3 Base Lower Base Address — The lower 32-bits of the address for the first queue entry in Receive Queue 3.



13.6.36 Receive Queue Upper Base Address Register 3 — RQUBAR3

The Receive Queue Upper Base Address Register 3 (RQUBAR3) represents the upper 4-bits of the address for the first queue entry in Receive Queue 3.

Table 543. Receive Queue Upper Base Address Register 3 — RQUBAR3

Intel XScale® processor internal bus address offset +0A9CH Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
31:4	00000000H	Reserved
3:0	0H	Receive Queue 3 Base Upper Base Address — The upper 4-bits of the address for the first queue entry in Receive Queue 3.



13.6.37 IMU Test and Set Registers — IMUTSR[0:511]

There are five hundred and twelve (512) 8-bit IMU Test and Set Registers (IMUTSRx). Each 8-bit entry implements 2 bits, one per Intel XScale® processor. Bit 0 corresponds to Intel XScale® processor 0 and bit 1 corresponds to Intel XScale® processor 1. The valid values of an IMUTSRx are 00_2 , 01_2 , and 10_2 . An Intel XScale® processor can only read the other Intel XScale® processor's bit. For example, an Intel XScale® processor can only affect the value of its bit by one of the following events:

- A read performed by an Intel XScale® processor to an IMUTSRx register when both bits are cleared will set that particular Intel XScale® processor bit in the IMUTSRx.
- A write of 0_2 to an IMUTSRx bit by an Intel XScale® processor will only clear its corresponding bit in the IMUTSRx register.

When an Intel XScale® processor reads an IMUTSRx register and both bits are cleared, the value returned to the Intel XScale® processor will be the current value of the bits (00_2 in this case), followed by the hardware setting that particular Intel XScale® processor bit. For example, if Intel XScale® processor 1 reads an IMUTSRx register and both bits are cleared, 00_2 will be returned as the value read and the hardware will set bit 1, which will result in bits[1:0] of the IMUTSRx equal to 10_2 .

When an IMUTSRx contains a non-zero value, and any of the Intel XScale® processor reads that particular register, the value of the bits will not be affected and the non-zero value will be returned to the Intel XScale® processor.

Note: When an Intel XScale® processor attempts a write of 1_2 to its bit in an IMUTSRx register, the bit will not be affected.

Note: Although the bits' attributes are defined as Read/Write, writing a 1_2 to a bit does not have any effect.

Warning: The IMU Test and Set Registers must be accessed one byte at a time. If multiple bytes are accessed in one transaction, the result will be unpredictable.



13.6.37.1 Endian Mode Support

When accessing the test and set register array, the IMU allows any of the Intel XScale® processors, regardless of their endian mode settings, to access the same data byte. Note that in little endian mode, the Intel XScale® processor drives byte for address 0 on the least significant byte lane of the internal data bus (i.e., byte lane 0 per 32-bit bus), and in big endian mode the Intel XScale® processor drives byte for address 0 on the most significant byte lane of the internal data bus (i.e., byte lane 3 of per 32-bit bus).

The test and set register array is organized in little endian order in the IMU regardless of the Intel XScale® processor endian modes. When the IMU test and set register array is being accessed by an Intel XScale® processor, the IMU detects the endian mode setting of that particular core. If the Intel XScale® processor is running in big endian mode, the IMU would dynamically byte-swap the aligned DWORD within which the data byte is contained. Refer to Figure 120. Otherwise, if the Intel XScale® processor is running in little endian mode, the IMU would simply keep the bytes in their respective byte lanes. Refer to Figure 121.

Figure 120. IMU handling Intel XScale® Processor running in Big Endian Mode

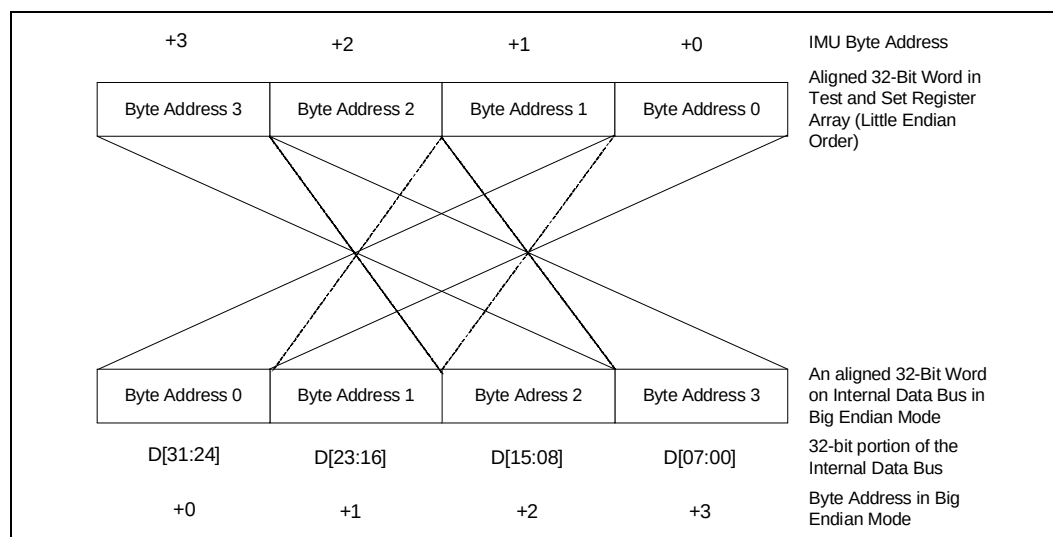


Figure 121. IMU handling Intel XScale® Processor running in Little Endian Mode

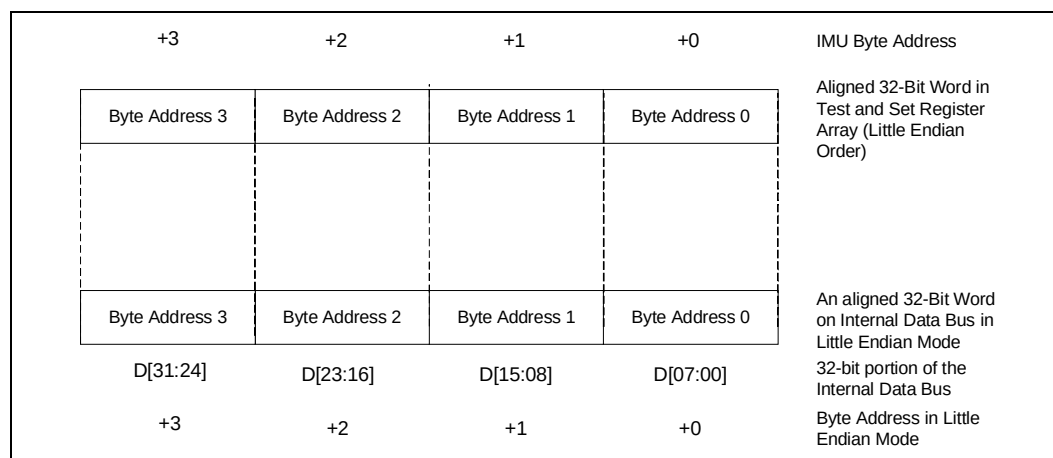




Table 544. IMU Test and Set Registers — IMUTSR[0:511]

Intel XScale® processor internal bus address offset IMUTSR0 +0B00H through IMUTSR511 +0CFFH Attribute Legend: RW = Read/Write, RV = Reserved, RC = Read Clear, PR = Preserved, RO = Read Only, RS = Read/Set, NA = Not Accessible		
Bit	Default	Description
07:02	00H	Reserved.
01	0 ₂	Intel XScale® processor 1 bit — This bit is only affected by Intel XScale® processor 1 transactions, and is a read-only bit for the other core (Intel XScale® processor 0). This bit is set when Intel XScale® processor 1 reads this register and both bit 0 and bit 1 are cleared. This bit is cleared when Intel XScale® processor 1 writes a 02 to this bit. Writing a 12 does not have any effect to this bit. Note: Both bits 1 and 0 cannot be 112 at any given time.
00	0 ₂	Intel XScale® processor 0 bit — This bit is only affected by Intel XScale® processor 0 transactions, and is a read-only bit for the other core (Intel XScale® processor 1). This bit is set when Intel XScale® processor 0 reads this register and both bit 0 and bit 1 are cleared. This bit is cleared when Intel XScale® processor 0 writes a 02 to this bit. Writing a 1b does not have any effect to this bit. Note: Both bits 1 and 0 cannot be 11b at any given time.



14.0 SMBus Interface Unit

This chapter describes the SMBus (System Management Bus) interface unit, including the operation modes and setup. Throughout this manual, this peripheral is referred to as the SMBus unit.

14.1 Overview

The SMBus Interface Units allows the Intel® 81341 and 81342 I/O Processors (81341 and 81342) to serve as a slave device residing on the SMBus. The SMBus is a two-pin interface. **SMBDAT** is the data pin for input and output functions and **SMBCLK** is the clock pin for reference and control of the SMBus.

The SMBus allows the system to interface to 81341 and 81342 for system management functions. The serial bus requires a minimum of hardware for an economical system to relay status and reliability information of the 81341 and 81342 to the system.

The SMBus Interface Unit is a peripheral device that resides on a 81341 and 81342 internal bus. Data is transmitted to and received from the SMBus via a buffered interface. Control and status information is relayed through a set of registers. Refer to the SMBus Specification for complete details on SMBus operation.

14.2 SMBus Interface

SMBus provides for full access to registers in 81341 and 81342 including configuration and memory-mapped registers. Systems so configured can use the SMBus to access the registers. 81341 and 81342 supports a slave-only SMBus mode.

- *System Management Bus Specification*, Revision 2.0 (SMBus) Compliant.
- Slave mode operation only.
- Full read/write access to configuration and memory-mapped register spaces in 81341 and 81342.

Table 545. SMBus Interface Pins

Signal	Pad Type
SMBCLK	SMBus Clock: Provides synchronous operation of the SMBus.
SMBDAT	SMBus Data: Used for data transfer and arbitration of the SMBus.
Total	2



14.3 System Management Bus Interface

This interface has no configuration registers associated with it. The SMBus address is set upon P_RST# by sampling the Peripheral Bus Interface Reset Strap inputs A[16:13]. When the pins are sampled, the resulting 81341 and 81342 address is stored in the Reset Strap Status Register and assigned as follows:

Bit	Value
7	1
6	1
5	A[16]
4	0
3	A[15]
2	A[14]
1	A[13]

The SMBus controller has access to all internal registers. It can perform reads and writes from all registers through the particular interface configuration space.



14.3.1 SMBus Controller

The 81341 and 81342 SMBus slave port interfaces to the configuration spaces of each ATU function, and also interfaces to the memory-mapped registers. This gives SM (server management) visibility into configuration space registers in the 81341 and 81342 ATUs 81341 and 81342.

14.3.1.1 SMBus Commands

The 81341 and 81342 supports six SMBus commands:

- Block Write
- Word Write
- Byte Write
- Block Read
- Word Read
- Bytes Read

Sequencing these commands initiates internal accesses to 81341 and 81342 configuration and memory-mapped registers. For high reliability, 81341 and 81342 also supports the optional Packet Error Checking feature (CRC-8) and is enabled or disabled with each transaction.

Every configuration and memory read or write first consists of an SMBus write sequence which initializes the Bus Number, Device, function number, memory address offset etc. The term sequence is used since these variables can be initialized by the SMBus master with a single block write or multiple word or byte writes. The last write in the sequence that completes the initialization performs the internal configuration/memory read or write. The SMBus master can then initiate a read sequence which returns the status of the internal read or write command and also the data in case of a read.

Each SMBus transaction has an 8-bit command driven by the master. The command encodes the following information:

Table 546. SMBus Command Encoding

Bit	Description
7	Begin: The Begin bit when set indicates the first transaction of the read or write sequence.
6	End: The End bit when set indicates the last transaction of the read or write sequence.
5	Memory/Configure: Indicate whether memory or configuration space is being accesses in this SMBus sequence. Value of '1' indicates memory and a value of '0' indicate configuration.
4	PEC Enable: Indicates that PEC is enabled when set. When set, each transaction in the sequence ends with an extra CRC byte. 81341 and 81342 would check for CRC on writes and generate CRC on reads.
3:2	Internal Command: 00 — Read DWord 01 — Write Byte 10 — Write Word 11 — Write Dword All access are naturally aligned to the access width. This field specifies the internal command to be issued by the SMBus slave logic to the 81341 and 81342
1:0	SMBus command: 00 — Byte 01 — Word 10 — Block 11 — Reserved This field specifies the SMBus command to be issued on the SMBus. This field is used as an indication of the length of transfer so that the slave knows when to expect the PEC packet (when enabled).



14.3.1.2 Initialization Sequence

All Configuration and memory reads and writes are accomplished through an SMBus write(s) and later followed by an SMBus read (for a read command). The SMBus write sequence is used to initialize the following for the configuration access:

- Bus Number (Bus Number is ignored on 81341 and 81342)
- Device/Function (Device Number is ignored on 81341 and 81342)
- 12-bit Register Number (in 2 separate bytes on SMBus)

Each of the parameters above is sent on SMBus in separate bytes. The register number parameter is initialized with two bytes and 81341 and 81342 ignores the most significant 4 bits of the second byte that initializes the register number.

For memory reads and writes, the write sequence initializes:

- a 19-bit memory address offset (in 3 separate bytes on SMBus)

On 81341 and 81342 memory transactions allow access to the Peripheral Memory-Mapped Registers only. All the Peripheral Memory-Mapped Registers on 81341 and 81342 are located in a 512-KByte contiguous memory space block that is relative to the PMMR Base Address Register (PMMRBAR). Refer to the Peripheral Registers Chapter for more details on the PMMRBAR register description. The 81341 and 81342 ignores the upper 5 bits of ADDR2 and the entire ADDR3 fields when memory transaction is selected. For example, an internal bus address is formed by concatenating the valid bits in the PMMRBAR and the 19 bits obtained from the SMBus in the ADDR2, ADDR1, and ADDR0 fields. Refer to [Table 548, “SMBus Interface Registers for Memory Space Access”](#) for the ADDR_x fields.

The initialization of the information can be accomplished through any combination of the supported SMBus write commands (Block, Word or Byte). The Internal Command field for each write should specify the same internal command every time (read or write). After all the information is set up, the last write (End bit is set) initiates an internal read or write command. On an internal read when the data is not available before the slave interface acknowledges this last write command (ACK), the slave does a “clock stretch” until the data returns to the SMBus interface unit. On a internal write, when the write is not complete before the slave interface acknowledges this last write command (ACK), the slave “clock stretches” until the write completes internally. When an error occurs (address error, target or master abort on the internal bus) during the internal access, the last write command receives a NACK.



14.3.2 SMBus Signaling

14.3.2.1 Overview

The SMBus interface includes a pair of signals: **SMBCLK** (clock) and **SMBDAT** (serial data). **SMBCLK** provides the timing mechanism for data transfers. The SMBus master always drives **SMBCLK**. The 81341 and 81342 may optionally extend **SMBCLK** low time by driving it low to meet setup timings on the SMBus.

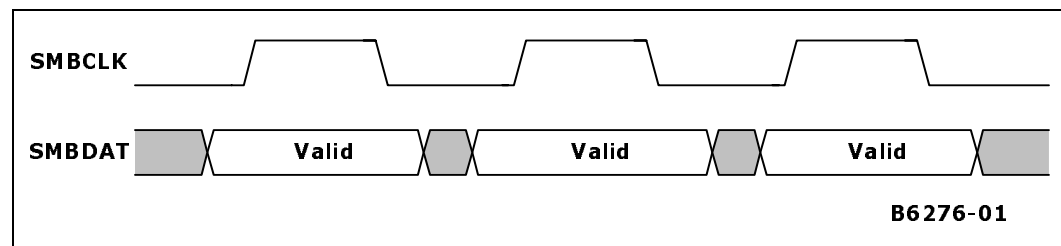
An initiator starts a transfer over the SMBus when it is free. Details of how initiators arbitrate are not described here. The current initiator communicates to the desired target through a unique 7-bit address to the target, sent MSb to LSb. All devices monitor the generated address after detecting the start condition. Once seven address bits are received, all targets compare the received address with their own and the target slave finds a match.

The next data bit from the initiator indicates the transfer direction. A value of '1' indicates that the target needs to transfer data to the initiator (read). Data transfers over SMBus are performed in 8-bit chunks. Data is transferred from MSb to LSb.

14.3.2.2 Waveforms

The timing relationship between **SMBDAT** and **SMBCLK** is defined such as the **SMBDAT** value must be valid through the duration of **SMBCLK** being in High state. The following diagram illustrates data transfer:

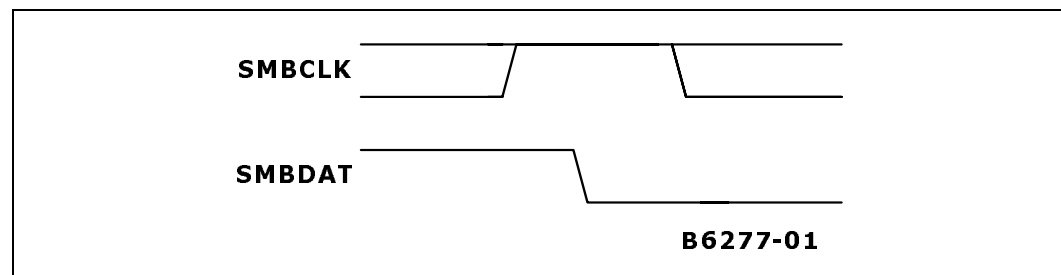
Figure 122. Basic SMBus Transfer Waveform



14.3.2.2.1 Start Phase

A start condition is generated when SMBus is idle to indicate that its state is changing to busy. The start condition occurs when **SMBDAT** transitions from High to Low while **SMBCLK** remains High. The SMBus protocol also allows a master to "Repeat Start", meaning that a new transfer is started by the same master, without a stop condition.

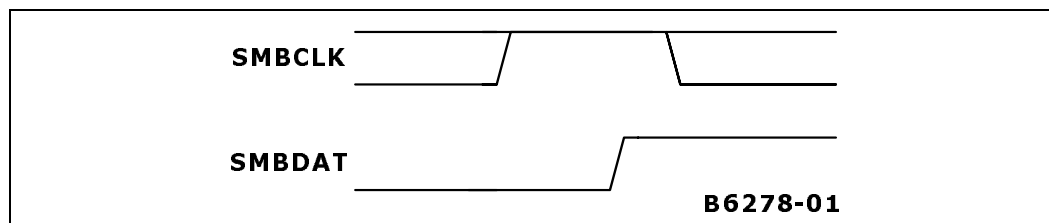
Figure 123. Start (S) / Repeat Start (Sr) Signaling



14.3.2.2.2 Stop Phase

A stop condition is generated when SMBus is busy to indicate that its state is changing to idle. The Stop condition occurs when **SMBDAT** transitions from Low to High while **SMBCLK** remains High.

Figure 124. Stop (P) Signaling



A stop bit can occur at any point in a data stream. It is not insured to occur after an ACK from a target (as later waveforms show). The 81341 and 81342 must be able to accept a stop condition at any time and clean up.

14.3.2.2.3 ACK/NACK

For every 8 bits of data transfer (including address and direction), the receiving agent must respond with ACK or NACK. An ACK is requires **SMBDAT** = 0 during **SMBCLK** = 1. A NACK requires **SMBDAT** = 1 during **SMBCLK** = 1 as shown below.

Figure 125. ACK (A) Signaling

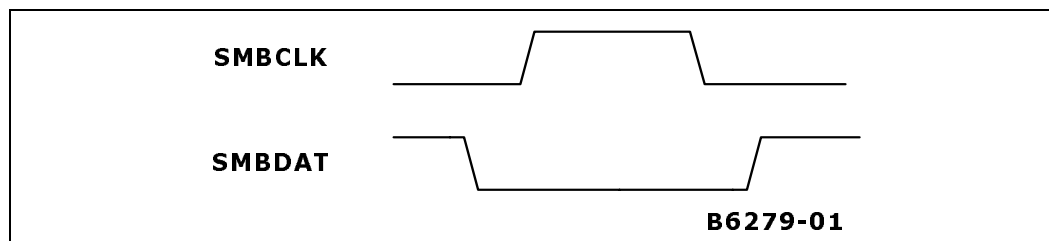
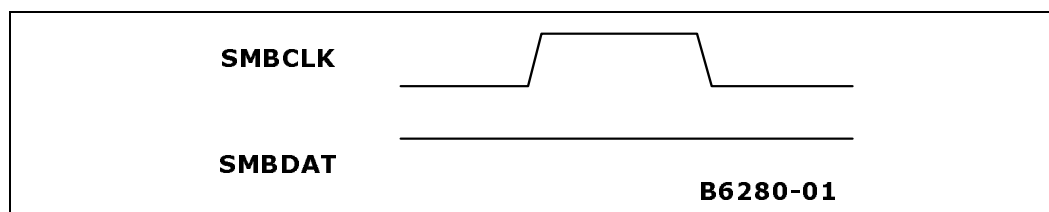


Figure 126. NACK (N) Signaling



During a write cycle, the 81341 and 81342 must drive an ACK after the address/direction phase, and after the data phase. During a read cycle, the 81341 and 81342 must drive an ACK/NACK after the address/direction phase, and (when ACKed) the initiator must drive an ACK/NACK after the 81341 and 81342 returns its 8 bits of data.

14.3.2.2.4 Wait States

The receiver (initiator or target) can add wait states, after driving ACK for receiving the last byte, by driving the **SMBCLK** line low. Further data transfers are delayed until the receiver stops driving **SMBCLK** low. It is expected the 81341 and 81342 drives the **SMBCLK** line low after receiving data on writes until the write is complete, and after receiving the direction bit on reads until the read data is ready.



14.3.3 Architecture

The 81341 and 81342 SMBus register interface consists of a set of registers that are only accessible from the SMBus interface only and are shown in [Table 547](#) and [Table 548](#). These registers are used to issue commands for reading and writing configuration registers and memory locations on 81341 and 81342. [Table 547](#) shows that register format for accessing configuration space. [Table 548](#) shows the register format for accessing memory space. The command register provides a bit that indicates whether configuration or memory space ought to be accessed.

Table 547. SMBus Interface Registers for Configuration Space Access

Register	Name and Function
CMD	Command
BYTCNT	Byte Count
ADDR3	Bus Number ^a
ADDR2	Device/Function Number. ^b
ADDR1	Extended Register Number — only bits [3:0]. The extended register allows access to 4-KByte configuration space.
ADDR0	Register Number — offset into function configuration space
DATA3	Data[31:24] — fourth byte of data.
DATA2	Data[23:16] — third byte of data.
DATA1	Data[15:8] — second byte of data.
DATA0	Data[7:0] — first byte of data.
STS	Status, only for reads.

a. The ADDR3 field is ignored as Bus Number is not applicable for 81341 and 81342.

b. Only ADDR2[2:0] are used to select a Function Number. ADDR2[7:3] are ignored as Device Number is not applicable to 81341 and 81342.

Table 548. SMBus Interface Registers for Memory Space Access

Register	Name and Function
CMD	Command
BYTCNT	Byte Count
ADDR3	Destination Memory. ^a
ADDR2	Address Offset — bits[23:16]. ^b
ADDR1	Address Offset — bits[15:8].
ADDR0	Address Offset — bits[7:0].
DATA3	Data[31:24] — fourth byte of data.
DATA2	Data[23:16] — third byte of data.
DATA1	Data[15:8] — second byte of data.
DATA0	Data[7:0] — first byte of data.
STS	Status, only for reads.

a. The ADDR3 field is ignored on 81341 and 81342 as only the Memory-Mapped Register block are accessible as a memory space.

b. Only ADDR2[2:0] are used on 81341 and 81342. ADDR2[7:3] are ignored as the Memory-Mapped Register Block only occupies 512 KBytes of memory space.

All SMBus accesses to internal register space are initiated via a write to the command register (CMD). The command register indicates the access type (read or write) and whether the command is targeting the configuration spaces or memory-mapped registers. Any register writes received by the 81341 and 81342 while a command is already in progress receive a NAK to prevent spurious operation. The master is no longer expected to poll the CMD register to prevent overwriting the current command in progress prior to issuing further writes. The SMBus access is delayed by stretching



the clock until such time that the data is delivered. Note that per the SMBus specification, this cannot be longer than 25 ms. To set up an internal access, the command register write is followed by four ADDR byte writes. Depending on the type of access, these four bytes indicate either the Bus number, Device, Function, Extended Register Offset, and Register Offset, or the Memory-mapped region selected and the address within the region. The configuration type access utilizes the traditional bus number, device, function, and register offset; but in addition, also uses an extended register offset which expands the addressable register space from 256 bytes to 4 Kbytes. The memory-mapped type access redefines these bytes to be a memory-mapped region selection byte, a filler byte which today is all zeroes, and then the memory address within the region. Refer to the earlier tables, which display this information. Note that the filler byte is today not utilized but enforces that both types of accesses have the same number of address bytes, and allows for future expansion.

The Command Register (CMD) indicates the type and size of transfer. All configuration accesses from the SMBus port are initiated by writing to the Command register. While a command is in progress, all future writes or reads are NACKed by the 81341 and 81342 to avoid having registers overwritten while in use. There are two command size fields to allow for more flexibility on how the data payload is transferred, both internally and externally. Refer to the Command Register for more details on the size supported. The command register also provides a begin bit and an end bit. These two bits support the breaking of the SMBus transactions up into smaller transfers, by defining the start and finish of an overall transfer.

The 81341 and 81342 SMBus interface supports byte, word, or block transfer sizes. A byte size indicates that the data payload is transferred one byte at a time per SMBus transaction. A word size indicates that the data payload is transferred one word (2 bytes) at a time in a single SMBus transaction. A block size allows a variable size data payload to be transferred in a single SMBus transaction. When a block transfer is to be performed an 8-bit byte-count field follows the command register. This byte-count field provides the 81341 and 81342 target with the expected number of bytes to anticipate from the SMBus Master. For a block read transaction, the 81341 and 81342 returns a byte-count value to the master when returning read data. This provides the master with a byte-count to anticipate. Note that on 81341 and 81342, the block transfer size is limited to only a maximum of four bytes.



14.3.3.1 Data Transfer Examples

For Figure 127 through Figure 134, the following terminology is used:

S	Start Bit
Sr	Start Repeat Bit
W	Write Command
R	Read Command
A	Acknowledge
N	Retry / not Acknowledge
P	Stop Bit

Clear boxes indicate phases of the cycle driven by the initiator, and shaded boxes indicate phases of the cycle driven by the target.

14.3.3.2 Configuration and Memory Reads

81341 and 81342 supports only read dword to internal register space. All Configuration and memory reads are accomplished through an SMBus write(s) and later followed by an SMBus read to read the status and the read data. For SMBus read transactions, the last byte of data (or the PEC byte when enabled), is NACKed by the master to indicate the end of the transaction. The SMBus memory read command returns the status of the previous internal command and the data associated previous internal read command. The status field encoding is:

Table 549. SMBus Status Byte Encoding

Bit	Description
7	Reserved
6	Internal Address Error
5	Internal Master Abort
4	Internal Target Abort
3:1	Reserved
0	Successful

Examples of configuration and memory reads are shown in Figure 127 through Figure 132. For the definition of the diagram conventions below, refer to the *System Management Bus Specification*, Revision 2.0.

Figure 127. DWORD Configuration Read Protocol (SMBus Block Write/Block Read, PEC Enabled)

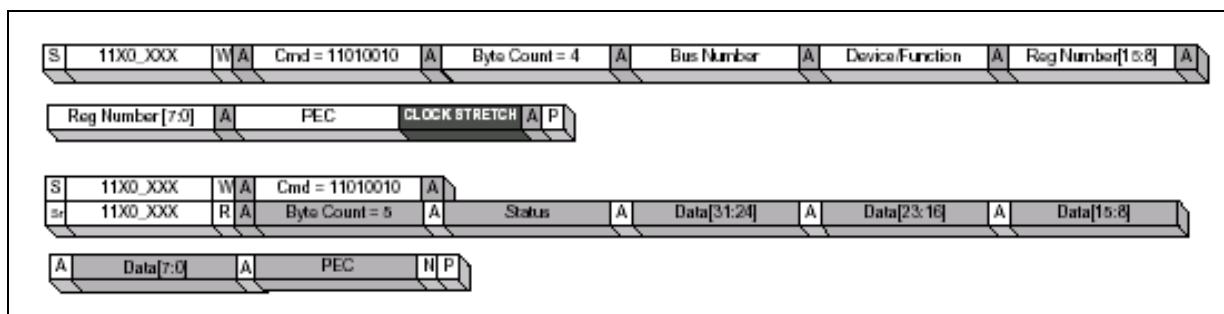


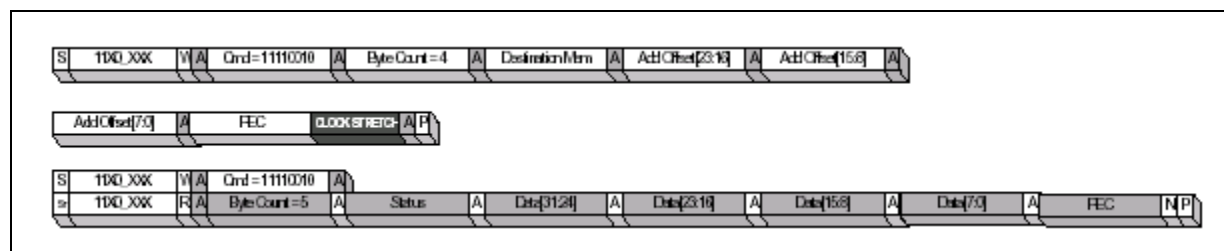
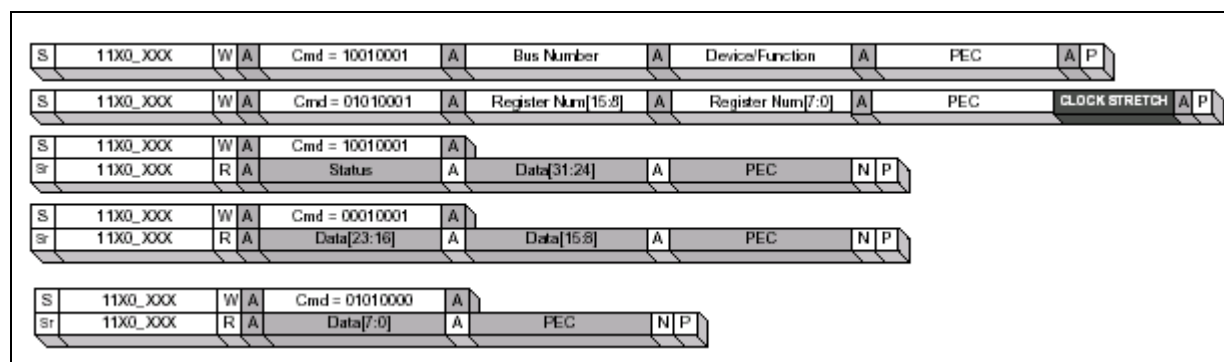
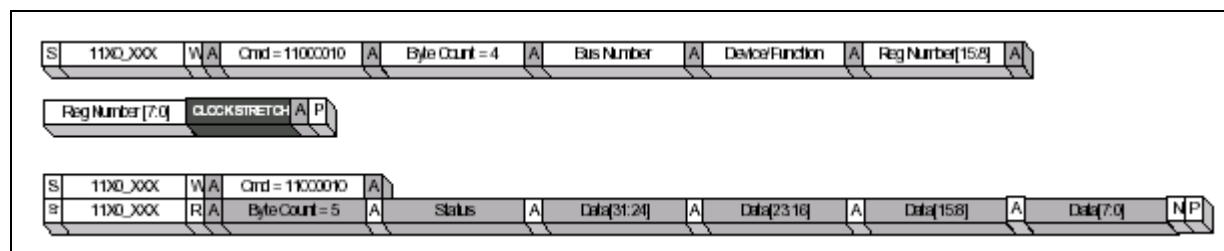
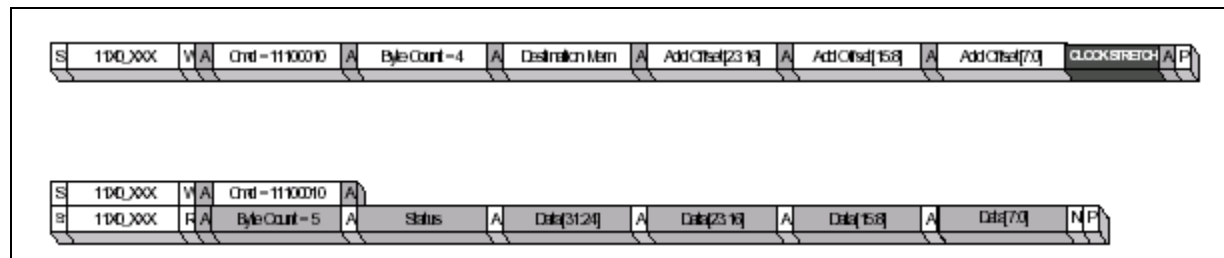
Figure 128. DWORD Memory Read Protocol (SMBus Block Write/Block Read, PEC Enabled)

Figure 129. DWORD Configuration Read Protocol (SMBus Word Write/Word Read, PEC Enabled)

Figure 130. DWORD Configuration Read Protocol (SMBus Block Write/Block Read, PEC Disabled)

Figure 131. DWORD Memory Read Protocol (SMBus Block Write/Block Read, PEC Disabled)




Figure 132. DWORD Configuration Read Protocol (SMBus Word Write/Word Read, PEC Disabled)

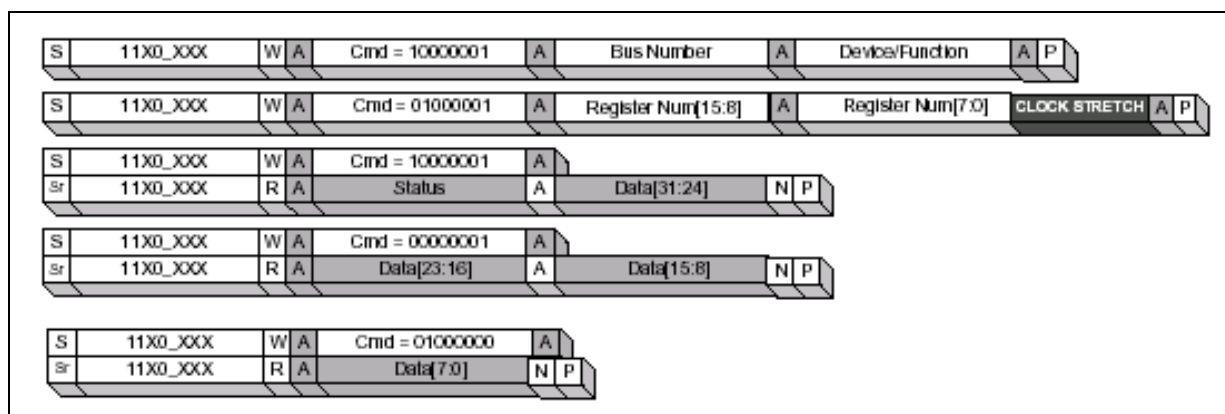


Figure 133. DWORD Memory Read Protocol (SMBus Word Write/(Word, Byte) Read, PEC Enabled)

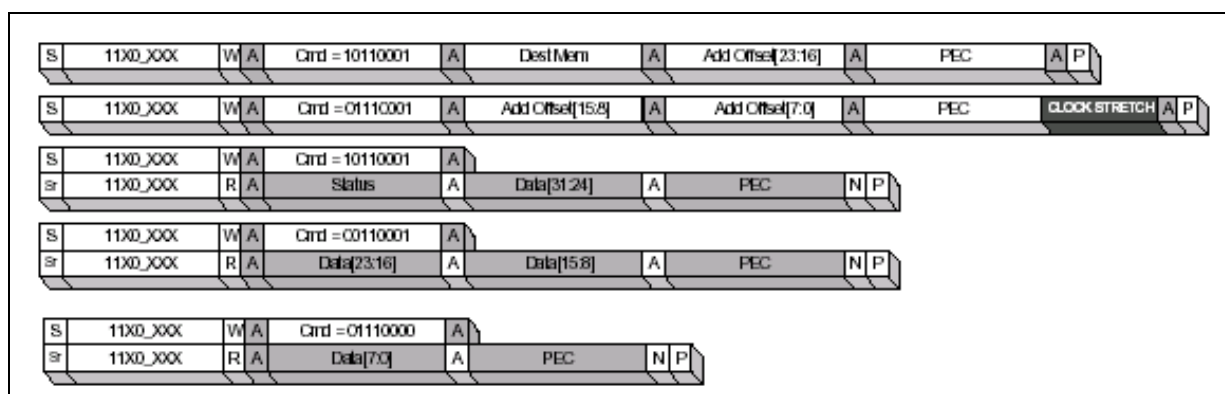
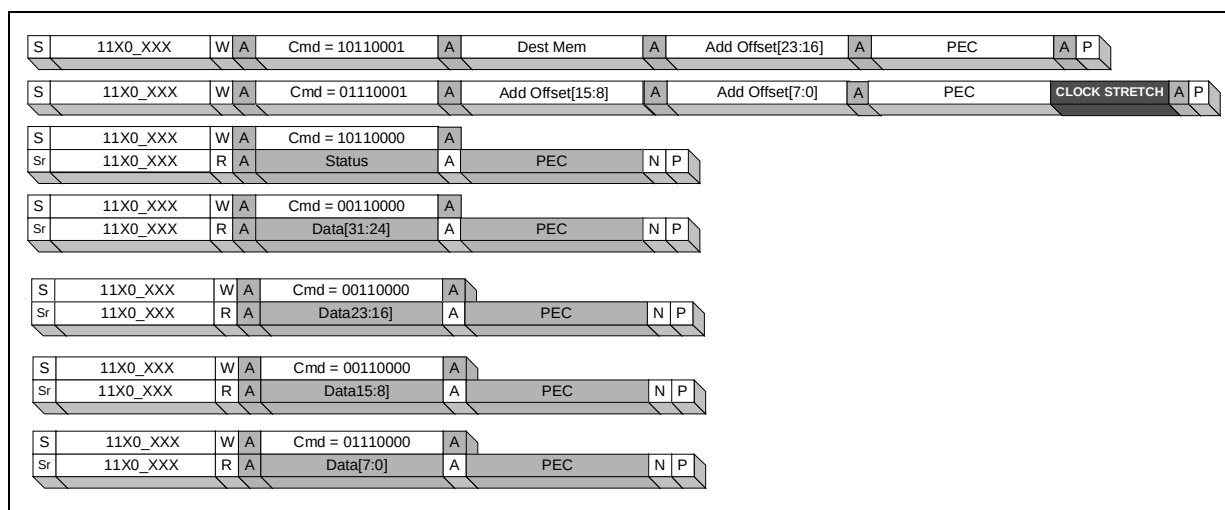


Figure 134. DWORD Memory Read Protocol (SMBus Word Write/Byte Read, PEC Enabled)



14.3.3.3 Configuration and Memory Writes

Configuration and memory writes are accomplished through a series of SMBus writes. As with reads, a write sequence is first used to initialize the Bus Number, Device, Function, and Register Number for the configuration access and the destination memory, address offset for the memory write. The writing of this information can be accomplished through any combination of the supported SMBus write commands (Block, Word or Byte).

Note: On SMBus, there is no concept of byte enables. Therefore, the Register Number written to the slave is assumed to be aligned to the length of the Internal Command. In other words, for a Write Byte internal command, the Register Number specifies the byte address. For a Write DWord internal command, the two least-significant bits of the Register Number are ignored. This is different from PCI where the byte enables are used to indicate the byte of interest.

After all the information is set up, the SMBus master initiates one or more writes which sets up the data to be written. The final write (End bit is set) initiates an internal configuration or memory write. The slave interface could potentially clock stretch the last data write until the write completes without error. When an error occurred, the SMBus interface NACKs the last write operation just before the stop bit. Examples of configuration writes are illustrated below. All the figures are with PEC Enabled. When PEC is disabled, there is no PEC byte in any of the sequences and the PEC enable bit in the command field is 0.

For the definition of the diagram conventions below, refer to the *System Management Bus Specification*, Revision 2.0.

Figure 135. DWORD Configuration Write Protocol (SMBus Block Write, PEC Enabled)

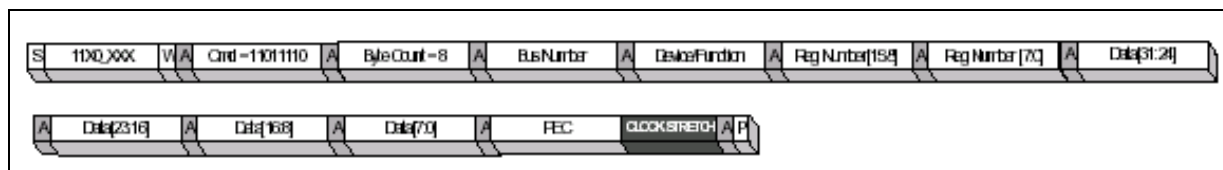


Figure 136. DWORD Memory Write Protocol (SMBus Word Write, PEC Enabled)

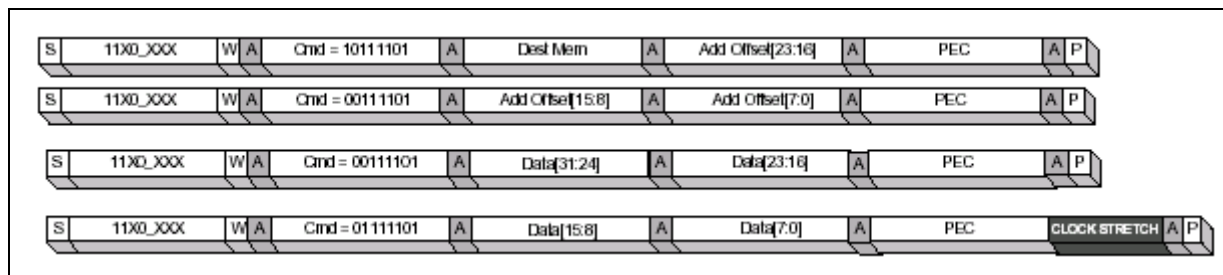
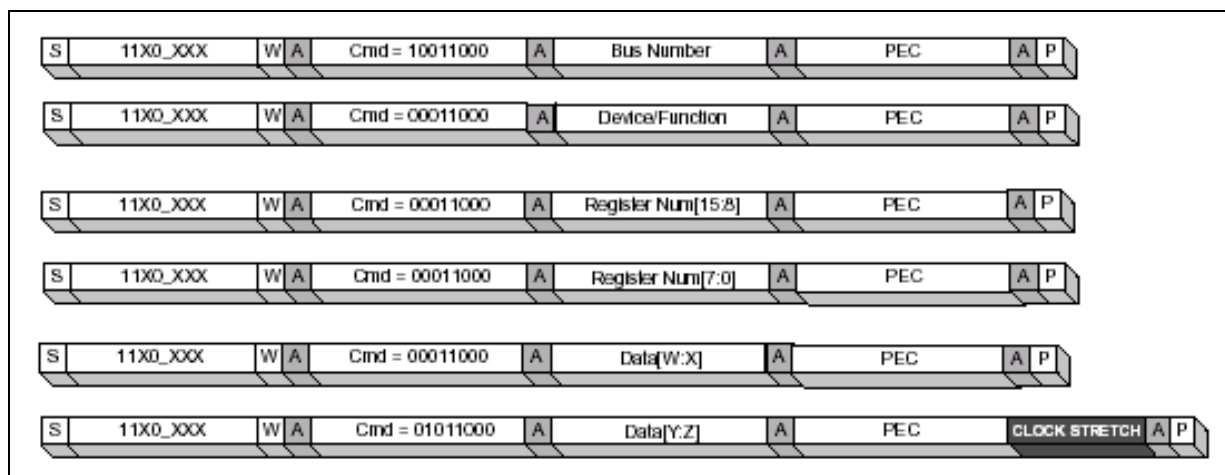



Figure 137. DWORD Configuration Write Protocol (SMBus Byte Write, PEC Enabled)




14.3.4 Error Handling

The SMBus slave interface handles two types of errors: internal address error and PEC. Internal address errors can occur, for example, when the SMBus request on the 81341 and 81342 internal bus fails due to an address parity error. This error manifests itself as a Not-Acknowledge (NACK) for the read or write command (End bit is set). Other internal errors include the read or write command receiving a master or target abort on the internal interface. When the master receives a NACK, the entire transaction should be reattempted.

When the master supports packet error checking (PEC) and the PEC enable bit in the command is set, then the PEC byte is checked in the slave interface. When the check indicates a failure, then the slave NACKs the PEC packet and does not issue the command on the internal interface.

An SMBus master must either do PEC on all transactions in a sequence or not do it at all; that is, it cannot turn on PEC in the middle of a sequence.

A PEC error in the middle of a sequence must be re-started from the beginning of the sequence; that is, the begin bit set.

14.3.5 SMBus Interface Reset

The master in two ways can reset the slave interface state machine in 81341 and 81342:

The master holds **SMBCLK** low for 25 ms cumulative. Cumulative in this case means that all the “low time” for **SMBCLK** is counted between the Start and Stop bit. When this totals 25 ms before reaching the Stop bit, the interface is reset.

The master holds **SMBCLK** continuously high for 50 ms.

Besides these, the SMBus interface in 81341 and 81342 is also reset on a P_RST#, WARM_RST# or an in-band warm reset from PCI Express*.



14.4 Register Definitions

This section provides a summary descriptions of all the SMBus registers. The SMBus Interface Unit has eight registers which are accessible from the SMBus interface only. Firmware cannot access these registers on 81341 and 81342.

Table 550. SMBus Register Summary

Section, Register Name, Acronym, Page
Section 551, "SMBus Controller Command Register — SM_CMD" on page 875
Section 552, "SMBus Controller Byte Count Register — SM_BC" on page 876
Section 553, "SMBus Controller ADDR3 Register — SM_ADDR3" on page 876
Section 554, "SMBus Controller ADDR2 Register — SM_ADDR2" on page 876
Section 555, "SMBus Controller ADDR1 Register Number — SM_ADDR1" on page 877
Section 556, "SMBus Controller ADDR0 Register Number — SM_ADDR0" on page 877
Section 557, "SMBus Controller Data Register — SM_DATA" on page 878
Section 558, "SMBus Controller Status Register — SM_STS" on page 878

14.4.1 SMBus Controller Command Register — SM_CMD

This is the Command Register. All accesses from the SMBus port are initiated by writing to this register. While a command is in progress, all future writes or reads are NACKed by the 81341 and 81342 to avoid having registers overwritten.

Table 551. SMBus Controller Command Register — SM_CMD

Bit	Reset	Description
07	0	Begin: The Begin bit when set indicates the first transaction of the read or write sequence.
06	0	End: The End bit when set indicates the last transaction of the read or write sequence.
05	0	Memory/Configure: Indicate whether memory or configuration space is being accesses in this SMBus sequence. Value of '1' indicates memory and a value of '0' indicate configuration.
04	0	PEC Enable: Indicates that PEC is enabled when set. When set, each transaction in the sequence ends with an extra CRC byte. 81341 and 81342 would check for CRC on writes and generate CRC on reads.
03:02	00	Internal Command: 00 — Read DWord 01 — Write Byte 10 — Write Word 11 — Write Dword All access are naturally aligned to the access width. This field specifies the internal command to be issued by the SMBus slave logic to the 81341 and 81342
01:00	00	SMBus command: 00 — Byte 01 — Word 10 — Block 11 — Reserved This field specifies the SMBus command to be issued on the SMBus. This field is used as an indication of the length of transfer so that the slave knows when to expect the PEC packet (when enabled).



14.4.2 SMBus Controller Byte Count Register — SM_BC

The SM_BC register indicates the number of bytes following the command field when performing a write or when setting up for a read. The byte count is also used when returning the data to indicate the following bytes including the status byte which is returned prior to the data. Note that the byte count is only transmitted for block type accesses on SMBus. SMBus word or byte accesses do not use the byte count register.

Table 552. SMBus Controller Byte Count Register — SM_BC

Bit	Reset	Description
07:00	00H	Byte Count: Indicates the number of bytes to anticipate for block size transfers. Not used for byte and word size transfers.

14.4.3 SMBus Controller ADDR3 Register — SM_ADDR3

The SM_ADDR3 register should be programmed with the Bus Number of the desired configuration register. The Status Register should be checked to make sure that there is not a command currently in progress, before writing to this register. Writing to this register when the 'Busy' bit in the Status Register is asserted has indeterminate effects. When accessing memory, the SM_ADDR3 register is ignored by 81341 and 81342.

Table 553. SMBus Controller ADDR3 Register — SM_ADDR3

Bit	Reset	Description
07:00	00H	ADDR3: Indicates the bus number to access when accessing configuration registers. Not used with memory access.

14.4.4 SMBus Controller ADDR2 Register — SM_ADDR2

This register should be programmed with the Device Number and Function Number of the desired configuration register. The Status Register should be checked to make sure that there is not a command currently in progress, before writing to this register. Writing to this register when the 'Busy' bit in the Status Register is asserted has indeterminate effects. When accessing memory, bits 0, 1 and 2 of the SM_ADDR2 register provides address bits [16,17,18] of the memory address offset. The upper 5 bits of SM_ADDR2 register are ignored by 81341 and 81342.

Table 554. SMBus Controller ADDR2 Register — SM_ADDR2

Bit	Reset	Description
07:03	00H	Device Number (DEV): Device number of device to access.
02:00	000	Function Number (FNC): Function number of device to access. For memory access, bit 0 represents bit 16 of the address offset.



14.4.5 SMBus Controller ADDR1 Register Number — SM_ADDR1

This register should be programmed with the upper address bits (bits [11:8]) of the Register Number of the desired configuration register for 4-KByte configuration space. 81341 and 81342 ignores bit [7:4] of this register. The Status Register should be checked to make sure that there is not a command currently in progress, before writing to this register. Writing to this register when the 'Busy' bit in the Status Register is asserted has indeterminate effects. When accessing memory space, ADDR1 provides bits [15:8] of the memory address offset.

Table 555. SMBus Controller ADDR1 Register Number — SM_ADDR1

Bit	Reset	Description
07:00	00H	Upper Number (UNUM): Indicates the upper 4 bits (bits [11:8]) of the register number to access for 4-KByte configuration space. Bits [7:4] of this registers is ignored by 81341 and 81342. For memory access, this register provides bits [15:8] of the memory address offset.

14.4.6 SMBus Controller ADDR0 Register Number — SM_ADDR0

This register should be programmed with the lower address bits (bits [7:0]) of the Register Number of the desired configuration register for 4-KByte configuration space. The Status Register should be checked to make sure that there is not a command currently in progress, before writing to this register. Writing to this register when the 'Busy' bit in the Status Register is asserted has indeterminate effects. When accessing memory space, SM_ADDR0 provides bits [7:0] of the memory address offset.

Table 556. SMBus Controller ADDR0 Register Number — SM_ADDR0

Bit	Reset	Description
07:00	00H	Lower Number (LNUM): Indicates the lower 8 bits (bits [7:0]) of the register number to access for 4-KByte configuration space. For memory access, this register provides bits [7:0] of the memory address offset.



14.4.7 SMBus Controller Data Register — SM_DATA

This register is used to read or write data to the desired Configuration Register.

At the completion of a Read command, this register contains the data from the selected configuration register. For reads the data register always returns 32 bits and is always aligned on a DWORD boundary.

Before issuing a write command this register should be written with the desired write data. For a byte, only the D[7:0] data is written to the desired configuration register. For a word write, only the D[15:0] data is written to the desired configuration register. The register number must be word aligned for word writes. For a DWORD write, all 32 bits of data are used. The register number must be DWORD aligned.

The Status Register should be checked to make sure that there is not a command currently in progress, before writing to this register. Writing to this register when the Busy bit in the Status Register is asserted, has indeterminate effects.

Table 557. SMBus Controller Data Register — SM_DATA

Bit	Reset	Description
31:24	00H	Byte 3 (B3): Data bits [31:24].
23:16	00H	Byte 2 (B2): Data bits [23:16].
15:08	00H	Byte 1 (B1): Data bits [15:8].
07:00	00H	Byte 0 (B0): Data bits [7:0].

14.4.8 SMBus Controller Status Register — SM_STS

The SM_STS Register provides the status of the internal transaction. For an SMBus read transaction, the data is preceded by a byte of status.

Table 558. SMBus Controller Status Register — SM_STS

Bit	Reset	Description
07	0	Reserved
06	0	Internal Address Error
05	0	Internal Master Abort
04	000	Internal Target Abort
03:01	0	Reserved
00	000	Successful

Notes:

1. An Address Error is signalled when an address parity is detected. The error is logged in the System Controller.



15.0 UARTs

This chapter describes the Universal Asynchronous Receiver/Transmitter (UART) serial ports. The Intel® 81341 and 81342 I/O Processors (81341 and 81342) UARTs are controlled via programmed I/O through memory-mapped registers.

15.1 Overview

Each asynchronous serial port supports all the functions of 16550 UART. Each UART performs 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 processor. The processor can read the complete status of a UART at any time during the functional operation. Available status information includes the type and condition of the transfer operations being performed by a UART and any error conditions (parity, overrun, framing, or break interrupt).

Each serial port can operate in either FIFO or non-FIFO mode. In FIFO mode, a 64-byte transmit FIFO holds data from the processor to be transmitted on the serial link and a 64-byte Receive FIFO buffers data from the serial link until read by the processor.

Each UART includes a programmable baud rate generator which is capable of dividing the input clock by divisors of 1 to $(2^{16}-1)$ and producing a 16X clock to drive the internal transmitter and receiver logic. Interrupts can be programmed to the user's requirements, minimizing the computing required to handle the communications link. Each UART operates in a polled or an interrupt driven environment which is selected by software.



The UART hardware is responsible for executing serial protocol communication and for providing the programming interface. The UART features include:

- Registers are compatible with the 16550 and 16750
- 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
- Baud-rate generator allows division of clock by 1 to $(2^{16} - 1)$ and generates an internal 16X clock; baud-rate can be manually or automatically programmed via auto-baud-rate detection circuitry
- Modem control functions (CTS#, RTS#)
- Autoflow capability controls data I/O without generating Interrupts:
 - RTS# (output) controlled by UART Receiver FIFO
 - CTS# (input) from modem controls UART transmitter
- 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 115kbps)
- False start bit detection
- 64-byte Transmit FIFO
- 64-byte Receive FIFO with programmable threshold
- Complete status reporting capability
- 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

15.1.1 Compatibility with 16550 and 16750

The UARTs can be programmed to be functionally compatible with industry standard 16550 and 16750. Each UART supports most of the 16550 and 16750 functions and has additional features, as listed below.

- DMA requests for transmit and receive data services
- NRZ encoding/decoding function
- 64 byte Transmit/Receive FIFO buffers
- Programmable Receive FIFO threshold
- Auto baud-rate detection
- Auto flow



15.2 Signal Descriptions

The name and description of external signals connected to a UART module are shown in Table 559.

Table 559. UART Signal Descriptions

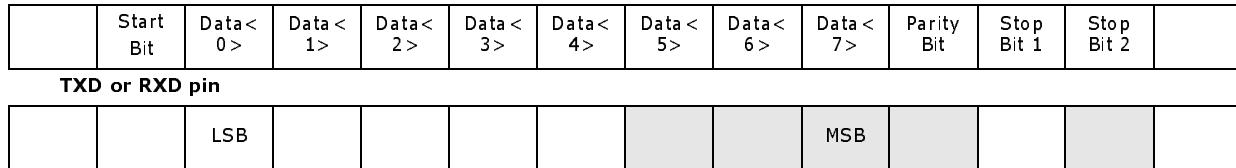
Name*	Type	Description
Ux_RXD	Input	SERIAL INPUT: Serial data input from device pin to the receive shift register.
Ux_TXD	Output	SERIAL OUTPUT: Composite serial data output to the communications link-peripheral, modem, or data set. The TXD signal is set to the MARKING (logic 1) state upon a Reset operation.
Ux_CTS#	Input	CLEAR TO SEND: When low, this pin indicates that the receiving UART is ready to receive data. When the receiving UART deasserts CTS# high, the transmitting UART should stop transmission to prevent overflow of the receiving UART's buffer. The CTS# signal is a modem-status input whose condition can be tested by the host processor or by the UART when in Autoflow mode as described below: Non-Autoflow Mode: When not in Autoflow mode, bit 4 (CTS) of the Modem Status register (MSR) indicates the state of CTS#. Bit 4 is the complement of the CTS# signal. Bit 0 (DCTS) of the Modem Status register indicates whether the CTS# input has changed state since the previous reading of the Modem Status register. CTS# has no effect on the transmitter. The user can program the UART to interrupt the processor when DCTS changes state. The programmer can then stall the outgoing data stream by starving the transmit FIFO or disabling the UART with the IER register. Note: When UART transmission is stalled by disabling the UART, the user does not receive an MSR interrupt when CTS# reasserts. This is because disabling the UART also disables interrupts. To get around this, the user can use Auto CTS in Autoflow Mode, or program the CTS# pin to interrupt. Autoflow Mode: In Autoflow mode, the UART Transmit circuitry checks the state of CTS# before transmitting each byte. When CTS# is high, no data is transmitted. See Section 15.4.7, UART x Modem Control Register for more information on Auto CTS mode.
Ux_RTS#	Output	REQUEST TO SEND: When low, this informs the remote device that the UART is ready to receive data. A reset operation sets this signal to its Inactive (high) state. LOOP mode operation holds this signal in its Inactive state. Non-Autoflow Mode: The RTS# output signal can be asserted by setting bit 1 (RTS) of the Modem Control register to a 1. The RTS bit is the complement of the RTS# signal. Autoflow Mode: RTS# is automatically asserted by the autoflow circuitry when the Receive buffer exceeds its programmed threshold. It is deasserted when enough bytes are removed from the buffer to lower the data level back to the threshold. See Section 15.4.7, UART x Modem Control Register for more information on Auto RTS mode.

Note: * "x" in signal name replaced with either "0" or "1" for UART-0 or UART-1 respectively.

15.3 Theory of Operation

The format of a UART data frame is shown in Figure 138.

Figure 138. Example UART Data Frame

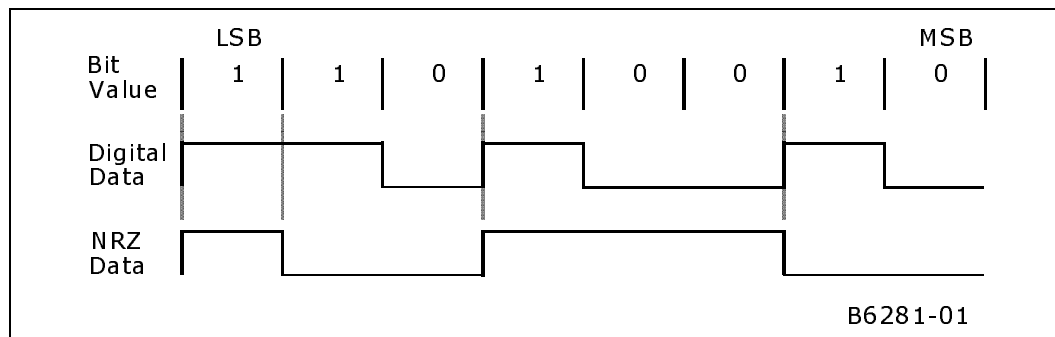


Shaded bits are optional and can be programmed by user. —>

Each data frame is between 7 bits and 12 bits long, depending on the size of data programmed and when parity and stop bits are enabled. The frame begins with a start bit that is represented by a high-to-low transition. Next, five to eight bits of data are transmitted, beginning with the least significant bit. An optional parity bit follows, which is set when even parity is enabled and an odd number of ones exist within the data byte; or when odd parity is enabled and the data byte contains an even number of ones. The data frame ends with one, one-and-one-half or two stop bits (as programmed by the user), which is represented by one or two successive bit periods of a logic one.

NRZ coding can be used by the UART to represent individual bit values. NRZ coding is enabled when Interrupt Enable Register (IER) bit-5 is set to high. A one is represented by a line transition and a zero is represented by no line transition. Figure 139 shows the NRZ coding of the data byte 8b 0100 1011. Note that the byte's LSB is transmitted first.

Figure 139. NRZ Bit Encoding Example – (0100 1011)



The unit is disabled upon reset, and users need to enable the unit by setting the UART Unit Enable bit (UUE, bit-6) of Interrupt Enable Register. When the unit is enabled, the receiver starts looking for the start bit of a frame; the transmitter sends data to the transmit data pin when there is data available in the transmit FIFO. Transmit data can be written to the FIFO before the unit is enabled. When the UART is disabled, the transmitter/receiver finishes the current byte being transmitted/received (when it is in the middle of transmitting/receiving a byte), and stops transmitting/receiving more data. Disabling the UART with the UUE bit does not clear transmission/reception with the original data.

Each UART has a Transmit FIFO and a Receive FIFO each holding 64 characters of data. There are two methods for moving data into/out of the FIFOs: **Interrupts and Polling**.



15.3.1 FIFO Interrupt Mode Operation

15.3.1.1 Receiver Interrupt

When the Receive FIFO and receiver interrupts are enabled (FCR[0]=1 and IER[0]=1), receiver interrupts occur as follows:

- The Receive Data Available Interrupt is asserted when the FIFO has reached its programmed trigger level. The interrupt is cleared when the FIFO drops below the programmed trigger level.
- The IIR Receive Data Available indication also occurs when the FIFO trigger level is reached, and like the interrupt, the bits are cleared when the FIFO drops below the trigger level.
- The Data Ready bit (DR in LSR register) is set to 1 as soon as a character is transferred from the shift register to the Receive FIFO. This bit is reset to 0 when the FIFO is empty.

15.3.1.2 Transmit Interrupt

When the transmitter FIFO and transmitter interrupt are enabled (FCR[0]=1, IER[1]=1), transmit interrupts occur as follows:

- When the Flow Control Register Transmitter Interrupt Level (TIL) bit (FCR[3]) is clear (0), The Transmit Data Request interrupt occurs when the transmit FIFO is half empty or more than half empty. The interrupt is cleared when the data level exceeds the half-empty mark. The interrupt is cleared as soon as the Transmit Holding Register is written or the IIR is read. 1 to 32 characters may be written to the transmit FIFO while servicing the interrupt when TIL=0.
- When the Flow Control Register Transmitter Interrupt Level (TIL) bit is set (1), The Transmit Data Request Interrupt occurs when the Transmit FIFO is empty. The interrupt is cleared as soon as the Transmit Holding Register is written or the IIR is read. 1 to 64 characters may be written to the Transmit FIFO while servicing the interrupt when TIL = 1.

Users could cause the UART Transmit FIFO to overflow when too many characters are written. FIFO underflow does not cause an error as the UART waits for the Transmit FIFO to be serviced.

15.3.2 Removing Trailing Bytes In Interrupt Mode

When the number of entries in the Receive FIFO is less than its trigger level, and no additional data is received, the remaining bytes are called trailing bytes. When the receive FIFO is being serviced by processor interrupts, trailing bytes need to be removed via the processor using the 16550 compliant character time-out interrupt: Time Out Detected (TOD) bit of Interrupt Identification Register. To enter this mode, users need to insure that the character time-out interrupt is enabled via IER[4].

To remove trailing bytes in Interrupt mode, the user must wait for the character time-out interrupt and then read all remaining bytes as indicated in the FIFO Occupancy Register (FOR), or read one byte at a time until the FIFO is empty. This can be determined by polling the Line Status Register bit 0 through programmed I/O.

15.3.2.1 Character Time-out Interrupt

When the Receiver FIFO and Receiver Time-out Interrupt are enabled, a character time-out interrupt (TOD) occurs to signal the presence of trailing bytes. The Interrupt is cleared and the timer is reset when a character is read from the Receiver FIFO. When a time-out Interrupt has not occurred, the time-out timer is reset after a new character is received or after the processor reads the Receiver FIFO.

When enabled via IER[4], a character time-out occurs under the following conditions:

- At least one character is in the FIFO.
- A character has not been received for the amount of time it takes to receive four or more characters at the current baud rate.
- The FIFO has not been read for the amount of time it takes to receive four or more characters

15.3.3 FIFO Polled Mode Operation

With the FIFOs enabled (TRFIFOE bit of FCR set to 1), clearing IER[7] and IER[4:0] puts the serial port in the FIFO polled mode of operation. Since the receiver and the transmitter are controlled separately, either one or both can be in the Polled Operation mode. In this mode, software checks Receiver and Transmitter status via the LSR. The processor polls the following bits for Receive and Transmit Data Service.

15.3.3.1 Receive Data Service

- Processor should check *Data Ready bit of LSR* which is set when 1 or more bytes remains in the Receive FIFO or Receive Buffer register (RBR).

15.3.3.2 Transmit Data Service

- Processor should check *Transmit Data Request bit of LSR* which is set when transmitter needs data.
- Processor can also check *Transmitter Empty bit of LSR*, which is set when the Transmit FIFO or Holding register is empty.



15.3.4 Autoflow Control

Autoflow Control uses the Clear-to-Send (**CTS#**) and Request-to-Send (**RTS#**) signals to automatically control the flow of data between the UART and external modem. When autoflow is enabled, the remote device is not allowed to send data unless the UART asserts **nRTS** low. When the UART deasserts **RTS#** while the remote device is sending data, the remote device is allowed to send one additional byte after **RTS#** is deasserted. An overflow could occur when the remote device violates this rule. Likewise, the UART is not allowed to transmit data unless the remote device asserts **CTS#** low. This feature increases system efficiency and eliminates the possibility of a Receive FIFO Overflow error due to long Interrupt latency.

Autoflow mode can be used in two ways: **Full autoflow**, automating both **CTS#** and **RTS#**, and **half autoflow**, automating only **CTS#**. Full Autoflow is enabled by writing a 1 to bits 1 and 5 of the Modem Control register (MCR). Auto-CTS-Only mode is enabled by writing a 1 to bit 5 and a 0 to bit 1 of the MCR register.

15.3.4.1 RTS Autoflow

When in full autoflow mode, **RTS#** is asserted when the UART FIFO is ready to receive data from the remote transmitter. This occurs when the amount of data in the Receive FIFO is below the programmable threshold value. When the amount of data in the Receive FIFO reaches the programmable threshold, **RTS#** is deasserted. It is asserted once again when enough bytes are removed from the FIFO to lower the data level below the threshold.

15.3.4.2 CTS Autoflow

When in Full or Half-Autoflow mode, **CTS#** is asserted by the remote receiver when the receiver is ready to receive data from the UART. The UART checks **CTS#** before sending the next byte of data and does not transmit the byte until **CTS#** is low. When **CTS#** goes high while the transfer of a byte is in progress, the transmitter completes this byte.

Note: Autoflow mode can be used only in conjunction with FIFO mode.

15.3.5 Auto-Baud-Rate Detection

Each UART supports auto-baud-rate detection. When enabled, UART counts the number of 33.334 MHz clock cycles within the start-bit pulse. This number is then written into the Auto-Baud-Count register (ACR) and used to calculate the baud rate. When ACR is written, an Auto-Baud-Lock Interrupt is generated (when enabled), and the UART automatically programs the Divisor Latch registers with the appropriate baud rate. When preferred, the processor can read the Auto-Baud-Count register using this information to program the Divisor-Latch registers with a baud rate calculated by the processor. After the baud rate has been programmed, the processor is responsible to verify that the predetermined characters (usually AT or at) are being received correctly.

When the UART programs Divisor Latch registers, users can choose between two auto-baud calculation methods: table- and formula-based. The method is selected via bit ABT of the Auto-Baud Control register (ABR). When the formula method is used, any baudrate allowed in Equation 34 can be programmed by the UART. This method works well for higher baud rates, but could possibly fail below 28.8 kbps when the remote transmitter's actual baud rate differs by more than one percent of its target.

Equation 34. Baud Rate Formula

$$\text{BaudRate} = \frac{33.334\text{MHz}}{(16 \times \text{Divisor})}$$

The table method is more immune to such errors since the table rejects uncommon baud rates and rounds to the common ones. The table method allows any baud rate in Equation 34 above 28.8 kbps. Below 28.8 kbps only baud rates of 19200, 14400, 9600, 4800, 1200, and 300 baud can be programmed by UART. Some typical values for Divisor and corresponding baud rates are provided in Table 560. Baud rates above 3600 baud require only Divisor Latch Low Register to be programmed, since Divisor Latch High Register would be 0.

Table 560. Divisor Values for Typical Baud Rates

Baud Rate	Divisor	UART Rate	Error
115.2K	18	115.74K	0.47%
57.6K	36	57.87K	0.47%
38.4K	54	38.58K	0.47%
33.6K	62	33.60K	0.01%
28.8K	72	28.94K	0.47%
19.2K	109	19.29K	0.47%
14.4K	145	14.47K	0.47%
9600	217	9645	0.47%
4800	434	4800	0.01%
3600	579	3600	0.01%
2400	868	2400	0.01%
1200	1736	1200	0.01%
600	3472	600	0.01%
300	6944	300	0.01%

When the baud rate is detected, auto-baud circuitry disarms itself by clearing bit ABE of the Auto-Baud Control register (ABR). To rearm the circuitry, ABE bit must be rewritten.

Note:

For the auto-baud-rate detection circuit to work correctly, the first data bit transmitted after the start bit must be a logic '1'. When a logic '0' is transmitted instead, the autobaud circuit counts the zero as part of the start bit, with an incorrect baud rate being programmed into DLL and DLH registers.



15.3.6 Manual Baud Rate Selection

Each UART contains a programmable Baud Rate Generator that is capable of taking the fixed input clock of 33.334 MHz and dividing it by any divisor from 1 to $(2^{16}-1)$. The baud-rate generator output frequency is 16 times the baud rate. Two 8-bit registers store the divisor in a 16-bit binary format. These Divisor Registers must be loaded during initialization to ensure proper operation. When both Divisor Latches are loaded with 0, the 16X output clock is stopped. Access to the Divisor latch can be done with a word write. [Equation 34](#) or [Table 560](#) are used by the programmer to select the Divisor Latch value for the desired baud rate.



15.4 Register Descriptions

There are 15 registers in each UART. The registers are all 32 bit registers, but only lower 8 bits have valid data. The 12 UART registers share eight address locations in the MMR address space. Table 561 shows the registers and their addresses as offsets of a base address. The base address for each UART is 32 bits and is internal bus address offset 2300H for UART 0, and 2340H for UART 1. Note that the state of the Divisor Latch Bit (DLAB), which is 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 561. UART Register Addresses as Offsets of a Base

UART Register Addresses	DLAB Bit Value	Name	Register Accessed
Base	0	UxRBR	UART x Receive BUFFER (read only)
Base	0	UxTHR	UART x Transmit BUFFER (write only)
Base + 04H	0	UxIER	UART x Interrupt Enable (R/W)
Base + 08H	X	UxIIR	UART x Interrupt I.D. (read only)
Base + 08H	X	UxFCR	UART x FIFO Control (write only)
Base + 0CH	X	UxLCR	UART x Line Control (R/W)
Base + 10H	X	UxMCR	UART x Modem Control (R/W)
Base + 14H	X	UxLSR	UART x Line Status (Read only)
Base + 18H	X	UxMSR	UART x Modem Status (Read only)
Base + 1CH	X	UxSPR	UART x Scratch Pad (R/W)
Base	1	UxDLL	UART x Divisor Latch (Low Byte, R/W)
Base + 04H	1	UxDLH	UART x Divisor Latch (High Byte, R/W)
Base + 24H	X	UxFOR	UART x FIFO Occupancy Register (R/W)
Base + 28H	X	UxABR	UART x Autobaud Control Register (R/W)
Base + 2CH	X	UxACR	UART x Autobaud Count Register (read only)

Table 562. UART Unit Registers

Section, Register Name, Acronym, page
Section 15.4.1, "UART x Receive Buffer Register" on page 890
Section 15.4.2, "UART x Transmit Holding Register" on page 890
Section 15.4.3, "UART x Interrupt Enable Register" on page 891
Section 15.4.4, "UART x Interrupt Identification Register" on page 892
Section 15.4.5, "UART x FIFO Control Register" on page 894
Section 15.4.6, "UART x Line Control Register" on page 896
Section 15.4.7, "UART x Modem Control Register" on page 898
Section 15.4.8, "UART x Line Status Register" on page 900
Section 15.4.9, "UART x Scratchpad Register" on page 904
Section 15.4.10, "Divisor Latch Registers" on page 905
Section 15.4.11, "UART x FIFO Occupancy Register" on page 906
Section 15.4.12, "UART x Auto-Baud Control Register" on page 907
Section 15.4.13, "UART x Auto-Baud Count Register" on page 908

**Table 563. UART Register MMR Addresses**

UART Register Addresses	DLAB Bit Value	Name	Register Accessed
+2300H	0	U0RBR	UART 0 Receive BUFFER (read only)
	0	U0THR	UART 0 Transmit BUFFER (write only)
+2304H	0	U0IER	UART 0 Interrupt Enable (R/W)
+2308H	X	U0IIR	UART 0 Interrupt I.D. (read only)
	X	U0FCR	UART 0 FIFO Control (write only)
+230CH	X	U0LCR	UART 0 Line Control (R/W)
+2310H	X	U0MCR	UART 0 Modem Control (R/W)
+2314H	X	U0LSR	UART 0 Line Status (Read only)
+2318H	X	U0MSR	UART 0 Modem Status (Read only)
+231CH	X	U0SPR	UART 0 Scratch Pad (R/W)
+2300H	1	U0DLL	UART 0 Divisor Latch (Low Byte, R/W)
+2304H	1	U0DLH	UART 0 Divisor Latch (High Byte, R/W)
+2324H	X	U0FOR	UART 0 FIFO Occupancy Register (R/W)
+2328H	X	U0ABR	UART 0 Autobaud Control Register (R/W)
+232CH	X	U0ACR	UART 0 Autobaud Count Register (read only)
+2340H	0	U1RBR	UART 1 Receive BUFFER (read only)
	0	U1THR	UART 1 Transmit BUFFER (write only)
+2344H	0	U1IER	UART 1 Interrupt Enable (R/W)
+2348H	X	U1IIR	UART 1 Interrupt I.D. (read only)
	X	U1FCR	UART 1 FIFO Control (write only)
+234CH	X	U1LCR	UART 1 Line Control (R/W)
+2350H	X	U1MCR	UART 1 Modem Control (R/W)
+2354H	X	U1LSR	UART 1 Line Status (Read only)
+2358H	X	U1MSR	UART 1 Modem Status (Read only)
+235CH	X	U1SPR	UART 1 Scratch Pad (R/W)
+2340H	1	U1DLL	UART 1 Divisor Latch (Low Byte, R/W)
+2344H	1	U1DLH	UART 1 Divisor Latch (High Byte, R/W)
+2364H	X	U1FOR	UART 1 FIFO Occupancy Register (R/W)
+2368H	X	U1ABR	UART 1 Autobaud Control Register (R/W)
+236CH	X	U1ACR	UART 1 Autobaud Count Register (read only)





15.4.3 UART x Interrupt Enable Register

This register enables six types of interrupts which set a value in the Interrupt Identification register. Each of the six interrupt types can be disabled by clearing the appropriate bit of the IER register. Similarly, by setting the appropriate bits, selected interrupts can be enabled.

This register also has the control bits of the unit enable and NRZ coding enable. The use of bit 7 to bit 4 is different from the register definition of standard 16550.

Note: A global interrupt enable/disable exists in the Modem Control Register bit 3 (IE). After reset, this bit must be set or no interrupts occurs, regardless of the state of the IER bits. See [Section 15.4.7, "UART x Modem Control Register"](#) on page 898.

Table 566. UART x Interrupt Enable Register - (UxIER)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address						Attribute Legend:		RW = Read/Write
0	+2304H (DLAB=x)						RV = Reserved		RC = Read Clear
1	+2344H (DLAB=x)						PR = Preserved		RO = Read Only
							RS = Read/Set		NA = Not Accessible
Bit	Default	Description							
31:8	00 0000h	Reserved							
7	0 ₂	Preserved							
6	0 ₂	UART Unit Enable (UUE): 0 = the unit is disabled 1 = the unit is enabled							
5	0 ₂	NRZ coding Enable (NRZE): 0 = NRZ coding disabled 1 = NRZ coding enabled							
4	0 ₂	Receiver Time Out Interrupt Enable: (RTOIE) 0 = Receiver data Time out Interrupt disabled 1 = Receiver data Time out Interrupt enabled							
3	0 ₂	Modem Interrupt Enable (MIE): 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled							
2	0 ₂	Receiver Line Status Interrupt Enable (RLSE): 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled							
1	0 ₂	Transmit Data request Interrupt Enable (TIE): 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled							
0	0 ₂	Receiver Data Available Interrupt Enable (RAVIE): 0 = Receiver Data Available (Trigger level reached) interrupt disabled 1 = Receiver Data Available (Trigger level reached) interrupt enabled							

15.4.4 UART x Interrupt Identification Register

The IIR register is read to determine the type and source of UART interrupts. To be 16550 compatible, the lower 4 bits (0-3) of the IIR register are priority encoded as shown in [Table 568, “Interrupt Identification Register Decode” on page 893](#). When two or more interrupts represented by bits (0-3) occur, only the interrupt with the highest priority is displayed. The upper 4 bits, (4-7) are not priority encoded. These bits asserts/deasserts independently of the lower 4 bits.

Bit 0 (nIP) is used to indicate the existence of an interrupt in the priority encoded bits (0-3) of the IIR register. A low signal on this bit indicates an encoded interrupt is pending. When this bit is high, no encoded interrupt is pending, regardless of the state of the other 3 bits. IP# has no effect or association with the upper bits four bits (4-7) which assert/deassert independently of IP#.

In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels (listed in [Table 568, “Interrupt Identification Register Decode” on page 893](#)) 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.

Table 567. UART x Interrupt Identification Register - (UxIIR)

Unit #	Intel XScale® Core internal bus address	Attribute Legend:	RW = Read/Write
0	+2308H (DLAB=x)	RV = Reserved	RC = Read Clear
1	+2348H (DLAB=x)	PR = Preserved	RO = Read Only
		RS = Read/Set	NA = Not Accessible
Bit	Default	Description	
31:8	00 0000h	Reserved	
7:6	00 ₂	FIFO Mode Enable Status (FIFOES[1:0]): 00 = Non-FIFO mode is selected 01 = Reserved 10 = Reserved 11 = FIFO mode is selected (TRFIFOE = 1)	
5	0 ₂	Reserved	
4	0 ₂	Autobaud Lock (ABL) 0 = Autobaud circuitry has not programmed Divisor Latch registers (DLL/DLH) 1 = Divisor Latch registers (DLL/DLH) programmed by autobaud circuitry	
3	0 ₂	Time Out Detected (TOD): 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)	
2:1	0 ₂	Interrupt Source Encoded (IID[1:0]): indicates a Modem Status Interrupt when the IP# bit is low. When IP# bit is high, there is no Interrupt. 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	1 ₂	Interrupt Pending (IP#): 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending	

**Table 568. Interrupt Identification Register Decode**

	Interrupt ID bits				Interrupt SET/RESET Function			
	3	2	1	0	Priority	Type	Source	RESET Control
IP#	0	0	0	1	-	None	No Interrupt is pending.	-
IID[11]	0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt.	Reading the Line Status Register.
IID[10]	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.
TOD	1	1	0	0	Second Highest	Character Timeout 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.
IID[01]	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 (when the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (when the source of the interrupt) or writing to the Transmitter FIFO.
IID[00]	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.
Non Prioritized Interrupts								
ABL	4				None	Autobaud Lock Indication	Autobaud circuitry has locked onto the baud rate.	Reading the IIR Register



15.4.5 UART x 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.

Table 569. UART x FIFO Control Register - (UxFCR) (Sheet 1 of 2)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address						Attribute Legend:		RW = Read/Write
0	+2308H (DLAB=x)						RV = Reserved		RC = Read Clear
1	+2348H (DLAB=x)						PR = Preserved		RO = Read Only
							RS = Read/Set		NA = Not Accessible
Bit	Default	Description							
31:8	00 0000h	Reserved							
7:6	00 ₂	Interrupt Trigger Level (ITL): When the number of bytes in the receiver 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							
5:4	0 ₂	Preserved							
3	0 ₂	Transmitter Interrupt Level (TIL): Setting TIL causes Transmitter Interrupts to occur when the Transmit FIFO is empty. Clearing TIL causes Transmitter Interrupts to occur when the Transmit FIFO is half empty. 0 = Interrupt when FIFO is half empty 1 = Interrupt when FIFO is empty							



Table 569. UART x FIFO Control Register - (UxFCR) (Sheet 2 of 2)

Unit #	Intel XScale® Core internal bus address	Attribute Legend:	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
0	+2308 H (DLAB=x)	RV = Reserved PR = Preserved RS = Read/Set	
1	+2348 H (DLAB=x)		

Bit	Default	Description
2	0 ₂	Reset Transmitter FIFO (RESETTF): When set, the Transmitter FIFO counter is reset to clear all the bytes in the FIFO. The <i>TDRQ</i> bit of LSR is set generating a Transmitter Requests Data Interrupt IID field of IIR when the <i>TIE</i> bit in the IER register is set. The Transmitter Shift register is not reset; it completes the current transmission. Any transmit FIFO Service-Request Interrupts are cleared. Note: After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
1	0 ₂	Reset Receiver FIFO (RESETRF): When set, the receiver FIFO counter is reset to clear all the bytes in the FIFO. The DR bit in 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. Any Receive FIFO Service Request Interrupts are cleared. Note: After the FIFO is cleared, RESETRF is automatically reset to 0. 0 = no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	0 ₂	Transmit and Receive FIFO Enable (TRFIFOE): Enables/disables the transmitter and receiver FIFOs. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFOs are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFOs. When changing from FIFO mode to non-FIFO mode and vice versa, data is automatically cleared from the FIFOs. Any FIFO Service Request Interrupts are cleared when TRFIFOE is cleared. Note: This bit must be 1 when other bits in this register are written, or the other bits are not programmed. 0 = FIFOs are disabled 1 = FIFOs are enabled



15.4.6 UART x Line Control Register

In the Line Control Register, the system programmer specifies 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 LCR has bits for accessing the Divisor Latch registers and causing a Break condition. The programmer can also read the contents of the Line Control Register. The read capability simplifies system programming and eliminates the need for separate storage in system memory.

Table 570. UART x Line Control Register - (UxLCR) (Sheet 1 of 2)

Unit #		Intel XScale® Core internal bus address	
0		+230CH (DLAB=x)	
1		+234CH (DLAB=x)	
		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible	
Bit	Default	Description	
31:8	00 0000h	Reserved	
7	0 ₂	Divisor Latch register Access Bit (DLAB): This bit must be set (1) to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be clear (0) to access the Receiver Buffer, the Transmit-Holding Register, or the Interrupt-Enable Register. This bit does not have to be set when using autobaud. 0 = access Transmit Holding register (THR), Receive Buffer Register (RBR) and Interrupt Enable Register. 1 = access Divisor Latch Registers (DLL and DLH).	
6	0 ₂	Set break (SB): Causes a Break condition to the receiving UART. When SB is set (1), the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is clear (0). This bit acts only on the TXD pin and has no effect on the transmitter logic. In FIFO mode, wait for the transmitter to be idle (TEMT=1) to set and clear the break bit. 0 = no effect on TXD output. 1 = forces TXD output to 0 (space).	
5	0 ₂	Sticky Parity (STKYP): Can be used in multiprocessor communications. When PEN and STKYP are set (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. When EPS is 0, then the bit at the parity bit location is transmitted as a 1. In the receiver, when STKYP and PEN are 1, then the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. When the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt when line status interrupts were enabled. For example, when EPS is 0, the receiver expects the bit received at the parity bit location to be 1. When 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. When PEN = 0, STKYP is ignored. 0 = no effect on parity bit. 1 = Forces parity bit to be opposite of EPS bit value.	

**Table 570. UART x Line Control Register - (UxLCR) (Sheet 2 of 2)**

Unit #	Intel XScale® Core internal bus address	Attribute Legend:	
0	+230CH (DLAB=x)	RV = Reserved	RW = Read/Write
1	+234CH (DLAB=x)	PR = Preserved	RC = Read Clear
		RS = Read/Set	RO = Read Only
			NA = Not Accessible
Bit	Default	Description	
4	0 ₂	Even Parity Select (EPS): When PEN is set (1) and EPS is clear (0), an odd number of logic ones is transmitted or checked in the data word bits and the parity bit. When PEN is set (1) and EPS is also set (1), an even number of logic ones is transmitted or checked in the data word bits and parity bit. When PEN = 0, EPS is ignored. 0 = sends or checks for odd parity. 1 = sends or checks for even parity.	
3	0 ₂	Parity Enable (PEN): When set (1), a parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The parity bit is used to produce an even or odd number of ones when the data word bits and the parity bit are summed.) 0 = no parity function. 1 = allows parity generation and checking.	
2	0 ₂	Stop bits (STB): This bit specifies the number of stop bits transmitted and received in each serial character. When STB is clear (0), one stop bit is generated in the transmitted data. When STB is set (1) when a 5-bit word length is selected via WLS[1:0], then 1 and one half stop bits are generated. When STB is set (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 = 1 stop bit 1 = 2 stop bits, except for 5-bit character then 1-1/2 bits	
1:0	00 ₂	Word Length Select (WLS[1:0]): Specifies 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	



15.4.7 UART x Modem Control Register

This register controls the interface with the modem or data set (or a peripheral device emulating a modem). The contents of the Modem Control register are described below:

Table 571. UART x Modem Control Register - (UxMCR) (Sheet 1 of 2)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2310H (DLAB=x)					RV = Reserved		RC = Read Clear	
1	+2350H (DLAB=x)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
31:6	000 0000h	Reserved							
5	0 ₂	Autoflow Control Enable (AFE): When set, autoflow control is enabled. Only auto-CTS is enabled when <i>RTS</i> is cleared in MCR while <i>AFE</i> is set. Both auto-CTS and auto-RTS are enabled when <i>AFE</i> and <i>RTS in MCR</i> are set. Auto-RTS is not enabled when <i>AFE</i> is not set regardless of the state of <i>RTS</i>. Autoflow automates the flow of data between the UART and the remote device. See Section 15.3.4, “Autoflow Control” on page 885 for more details. 0 = Auto-RTS and Auto-CTS are disabled 1 = Auto-CTS is enabled. IF <i>RTS in MCR</i> is also set, both auto-CTS and auto-RTS is enabled							
4	0 ₂	Loop back test mode (LOOP): This bit provides a local Loopback feature for diagnostic testing of the UART. In the Diagnostic mode, data that is transmitted is immediately received. This feature allows the processor to verify the UART Transmit and Receive data paths. The Transmit, Receive and Modem Control Interrupts are operational. The modem-control input CTS# is activated by MCR bit 1 instead of the modem-control input. A Break signal can also be transferred from the transmitter section to the receiver section in Loop-Back mode. When LOOP is set (1), the following occurs: • The TXD (transmitter output) pin is set to a logic-1 state. • The RXD (receiver input) pin is disconnected. • The output of the Transmitter Shift register is “looped back” into the Receiver-Shift register input. • The modem-control input CTS# is disconnected from the pins and the modem-control output pin RTS# is forced to the inactive state. The <i>RTS</i> bit of the Modem Control register is connected to bits of the Modem Status register bits. Flow control can be tested; when autoflow is enabled the <i>RTS</i> bit of the Modem Control Register has no effect on the CTS# input as RTS# is asserted by the autoflow logic.: • <i>RTS</i> = 1 forces <i>CTS</i> to a 1 Note: Note: Coming out of the Loop-Back Test mode may result in unpredictable activation of the delta bit (bit 0) in the Modem Status register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. 0 = Normal UART operation 1 = Test mode UART operation							



Table 571. UART x Modem Control Register - (UxMCR) (Sheet 2 of 2)

IOP Attributes

Unit #	Intel XScale® Core internal bus address	Attribute Legend:	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
0	+2310 H (DLAB=x)	RV = Reserved	
1	+2350 H (DLAB=x)	PR = Preserved RS = Read/Set	

Bit	Default	Description
3	0 ₂	Interrupt Enable (IE): Global control all UART interrupts. 0 = interrupts disabled. 1 = interrupts enabled. NOTE: This bit is not valid when in Loopback mode.
2	0 ₂	Preserved.
1	0 ₂	Request to Send (RTS): Non-Autoflow mode: When not in Autoflow mode (AFE bit of MCR is clear), this bit controls the Request-to-Send (RTS#) output pin. 0 = RTS# pin is 1 1 = RTS# pin is 0 Autoflow mode: When in Autoflow mode (AFE bit of MCR is set), auto-RTS is enabled. RTS# behaves as follows: • Auto-RTS disabled. Autoflow works only with auto-CTS. • Auto-RTS enabled. Autoflow works with both auto-CTS and auto-RTS.
0	0 ₂	Reserved



This register provides status information to the processor concerning the data transfers. Bits 5 and 6 show information about the transmitter section. The remainder of the bits contain information about the receiver.

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. In FIFO mode, these three status bits are stored with each received character in the FIFO. LSR shows the status bits of the character at the bottom of the FIFO. When the character at the bottom of the FIFO has errors, the LSR error bits are set and are not cleared until software reads LSR, even when the character in the FIFO is read and a new character is now at the bottom of the FIFO.

Bits 1 through 4 are the error conditions that produce a Receiver Line Status Interrupt when any of the corresponding conditions are detected and the interrupt is enabled. These bits are not cleared by reading the erroneous byte from the FIFO or receive buffer. They are cleared only by reading LSR. In FIFO mode, the Line Status Interrupt occurs only when the erroneous byte is at the bottom of the FIFO. When the erroneous byte being received is not at the bottom of the FIFO, an interrupt is generated only after the previous bytes are read and the erroneous byte is moved to the bottom of the FIFO.

UART x Line Status Register - (UxLSR) (Sheet 1 of 3)

The diagram illustrates the mapping of PCI Attributes to Core internal bus addresses. It shows two rows of attributes, labeled 'PCI Attributes' on the left. The top row contains RV (Reserved), PR (Preserved), RS (Read/Set), RW (Read/Write), and RO (Read Only) attributes. The bottom row contains NA (Not Accessible) attributes. These attributes are mapped to core internal bus addresses, which are shown as a sequence of 32-bit words. The addresses are grouped by unit number (Unit # 0 and Unit # 1) and are labeled with their corresponding PCI Attribute values.

Unit #	Intel XScale® Core internal bus address	Attribute Legend:	RW = Read/Write RC = Read Clear PR = Preserved RO = Read Only NA = Not Accessible
0	+2314H (DLAB=x)		
1	+2354H (DLAB=x)		



Table 572. UART x Line Status Register - (UxLSR) (Sheet 2 of 3)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2314H (DLAB=x)					RV = Reserved		RC = Read Clear	
1	+2354H (DLAB=x)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
5	1 ₂	Transmit Data Request (TDRQ): Indicates that the UART is ready to accept data for transmission. The assertion of this bit causes the UART to issue an interrupt when the Transmit Data Request Interrupt Enable is set. In non-FIFO mode, the TDRQ bit is set (1) when a character is transferred from the Transmit-Holding register. The bit is cleared (0) concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set (1) when the FIFO is less than half full. It is cleared when the FIFO is more than half full. When more than 64 characters are loaded into the FIFO, the excess characters are lost. 0 = The UART is NOT ready to receive data for transmission. 1 = The UART is ready to receive data for transmission.							
4	0 ₂	Break Indicator (BI): Set (1) when the received data input is held in the Spacing (logic 0) state for longer than a full character transmission time (that is, the total time of <i>start</i> bit + <i>data</i> bits + <i>parity</i> bit + <i>stop</i> bits). The Break Indicator is reset (cleared to 0) when the processor reads the Line-Status register. In FIFO mode, only one Break character (equal to 0x00), is loaded into the FIFO regardless of the length of the Break condition. <i>BI</i> shows the Break condition for the character at the bottom of the FIFO, not the most recent character received. 0 = No break signal has been received. 1 = Break signal has been received.							
3	0 ₂	Framing Error (FE): Indicates that the received character did not have a valid stop bit. FE is set (1) when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). When the Line Control register had been set for two stop bits, the receiver does not check for a valid second stop bit. The FE indicator is reset when the processor reads the Line Status Register. The UART resynchronizes after a framing error by assuming that the framing error was due to the next start bit. Therefore 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. 0 = No Framing error. 1 = Invalid stop bit has been detected.							



Table 572. UART x Line Status Register - (UxLSR) (Sheet 3 of 3)

The diagram illustrates the Intel XScale Core internal bus address and attribute legend. It shows a 32-bit address bus with bits 31 down to 0. The address is divided into four 8-bit segments, each labeled with a hexadecimal value: 31, 28, 24, 20, 16, 12, 8, and 4. The address is also divided into four 8-bit segments, each labeled with a hexadecimal value: 31, 28, 24, 20, 16, 12, 8, and 4. The address is also divided into four 8-bit segments, each labeled with a hexadecimal value: 31, 28, 24, 20, 16, 12, 8, and 4. The address is also divided into four 8-bit segments, each labeled with a hexadecimal value: 31, 28, 24, 20, 16, 12, 8, and 4.

Attribute Legend:

- RW = Read/Write
- RC = Read Clear
- PR = Preserved
- RS = Read/Set
- RO = Read Only
- NA = Not Accessible

Unit #

- 0: +2314H (DLAB=x)
- 1: +2354H (DLAB=x)



Table 573. UART x 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). In addition to this current state information, the Modem Status register also provides change information. The change bit is set to a logic 1 when the control input from the Modem changes state. The change bit is reset to a logic 0 when the processor reads the Modem Status register.

Note: When the change bit (bit 0) is set to logic 1, a Modem Status interrupt is generated when bit 3 of the Interrupt Enable Register is set.

Table 574. UART x Modem Status Register - (UxMSR)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2318 H (DLAB=x)					RV = Reserved		RC = Read Clear	
1	+2358 H (DLAB=x)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
31:5	000 0000h	Reserved							
4	0 ₂	Clear to Send (CTS): This bit is the complement of the Clear to Send (CTS#) input. This bit is equivalent to bit RTS of the Modem Control register when LOOP in the MCR is set to 1. 0 = CTS# pin is 1 1 = CTS# pin is 0							
3:1	000 ₂	Reserved							
0	0 ₂	Delta Clear To Send (DCTS): 0 = No change in CTS# pin since last read of MSR 1 = CTS# pin has changed state							



15.4.9 UART x Scratchpad Register

This read/write register has no effect on the UART. It is intended as a scratchpad register for use by programmers.

Table 575. UART x Scratchpad Register - (UxSCR)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address						Attribute Legend:		RW = Read/Write
0	+231CH (DLAB=x)						RV = Reserved		RC = Read Clear
1	+235CH (DLAB=x)						PR = Preserved		RO = Read Only
							RS = Read/Set		NA = Not Accessible
Bit	Default	Description							
31:8	00 0000h	Reserved							
7:0	00h	No effect on UART functionality							



15.4.10 Divisor Latch Registers

The description of use for the Divisor Latch Registers are provided in [Section 15.3.5, Auto-Baud-Rate Detection](#) and [Section 15.3.6, Manual Baud Rate Selection](#). Refer to those sections for details on how to program these registers.

Bit DLAB in the LCR register must be set high before the Divisor Latch registers can be accessed.

A Divisor value of 0 in the Divisor Latch Register is not allowed. A value of 0 has the affect of disabling the UART. The reset value of the divisor is 02.

Table 576. UART x Divisor Latch Low Register - (UxDLL)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2300H (DLAB=1)					RV = Reserved		RC = Read Clear	
1	+2340H (DLAB=1)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
31:8	00 0000h	Reserved							
7:0	02h	Low byte compare value to generate baud rate							

Table 577. UART x Divisor Latch High Register - (UxDLH)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2304H (DLAB=1)					RV = Reserved		RC = Read Clear	
1	+2344H (DLAB=1)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
31:8	00 0000h	Reserved							
7:0	00h	High byte compare value to generate baud rate							



15.4.11 UART x FIFO Occupancy Register

This register shows the number of bytes currently remaining the Receive FIFO. It can be used by the processor to determine the number of trailing bytes to remove from the receive FIFO when the Character Time-out Interrupt is detected. Refer to [Section 15.3.2, “Removing Trailing Bytes In Interrupt Mode” on page 884](#). The FOR register is incremented once for each byte of data written to the Receive FIFO and decremented once for each byte read.

Table 578. UART x FIFO Occupancy Register - (UxFOR)

IOP Attributes																																
	PCI Attributes																															
Unit #	Intel XScale® Core internal bus address																Attribute Legend:								RW = Read/Write							
0	+2324H (DLAB=x)																RV = Reserved								RC = Read Clear							
1	+2364H (DLAB=x)																PR = Preserved								RO = Read Only							
																	RS = Read/Set								NA = Not Accessible							
Bit	Default	Description																														
31:7	000 0000h	Reserved																														
6:0	000 0000 ₂	FOR[6:0]: Number of bytes (0-63) remaining in the Receiver FIFO																														



15.4.12 UART x Auto-Baud Control Register

This read/write register has no effect on the UART. It is intended as a scratchpad register for use by programmers.

Table 579. UART x Auto-Baud Control Register - (UxABR)

IOP Attributes	31	28	24	20	16	12	8	4	0
	rv	rv	rv	rv	rv	rv	rv	rv	rv
PCI Attributes	na	na	na	na	na	na	na	na	na
	na	na	na	na	na	na	na	na	na
Unit #	Intel XScale® Core internal bus address					Attribute Legend:		RW = Read/Write	
0	+2328 H (DLAB=x)					RV = Reserved		RC = Read Clear	
1	+2368 H (DLAB=x)					PR = Preserved		RO = Read Only	
						RS = Read/Set		NA = Not Accessible	
Bit	Default	Description							
31:4	000 0000h	Reserved							
3	0 ₂	Auto-Baud Table (ABT): Directs the auto-baud circuitry within the UART to use a table when selecting the final baud rate programmed into the Divisor Latch registers and written into the Auto-Baud Count register (ACR). When a table is not used, any value allowed in Equation 34 on page 886 can be selected by the UART. See Section 15.3.5, "Auto-Baud-Rate Detection" on page 886 for more information on auto-baud. 0 = Formula used to calculate baud rates allowing all possible baud rates to be chosen by UART Equation 34 on page 886 . 1 = Table used to calculate baud rates which limits UART to choosing common baud rates.							
2	0 ₂	Auto-Baud UART Program (ABUP): Allows the UART to automatically program the Divisor Latch registers (DLL and DLH) when the auto-baud circuit has detected the baud rate, regardless of the state of the <i>divisor-latch access</i> bit (DLAB). For this to occur, the <i>auto-baud enable (ABE)</i> bit must be set. Clearing this bit allows the processor to read the Auto-Baud Count register (ACR) and determine the baud rate using its own algorithm rather than using the UARTs. 0 = Software programs Divisor Latch Registers. 1 = UART Programs Divisor Latch Registers							
1	0 ₂	Auto-Baud Lock Interrupt Enable (ABLIE): Enables the ABL Interrupt which occurs when the auto-baud circuit has detected the baud rate and written the value into the Auto-Baud Count register (ACR). The Divisor Latch registers can then be programmed by the processor or auto-baud circuitry as dictated by the state of the ABPP. 0 = Autobaud Lock Interrupt Disabled 1 = Autobaud Lock Interrupt Enabled							
0	0 ₂	Auto-Baud Enable (ABE): Enables the auto-baud circuitry within the UART. The circuitry counts the number of clocks in the <i>start</i> bit and writes this count into the Autobaud Count register (ACR). It then interrupts the processor when the <i>auto-baud lock interrupt-enable (ABLIE)</i> bit is set. It also automatically programs the Divisor Latch registers (DLL and DLH) when the <i>auto-baud UART program (ABUP)</i> bit is set. 0 = Autobaud Disabled 1 = Autobaud Enabled							



The Auto-Baud Count register stores the number of 33.334 MHz clock cycles within a *start* bit pulse. This value is then used by the processor or auto-baud circuitry within the UART to calculate the baud rate. When Auto-Baud mode and Auto-Baud Interrupts are enabled, the UART interrupt the processor with the Auto-Baud Lock Interrupt after it has written the count value into the ACR. The value is written regardless of the state of the *auto-baud UART program* bit.

Bit	Default	Description
31:16	0000h	Reserved
15:0	0000h	ACR[15:0]: Number of 33.334 MHz clock cycles within a start bit pulse.



16.0 I²C Bus Interface Units

This chapter describes the three I²C (Inter-Integrated Circuit) bus interface units, including the operation modes and setup. Throughout this manual, these peripherals are referred to as the I²C units.

16.1 Overview

The three I²C Bus Interface Units allows the Intel® 81341 and 81342 I/O Processors (81341 and 81342) to serve as a master and slave device residing on the I²C bus. The I²C bus is a serial bus developed by Philips* Corporation 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 81341 and 81342 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 81341 and 81342 subsystem to an external device.

The I²C Bus Interface Unit is a peripheral device that resides on a 81341 and 81342 internal bus. 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.

16.2 Theory of Operation

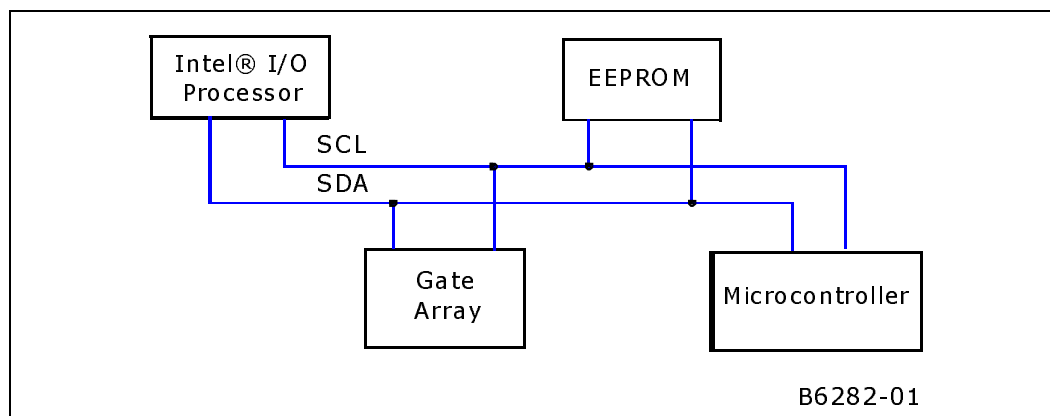
The I²C bus defines a serial protocol for passing information between agents on the I²C bus, using only a two pin interface. The interface consists of a Serial Data/Address (**SDA**) line and a Serial Clock Line (**SCL**). Each device on the I²C bus is recognized by a unique 7-bit address and can operate as a transmitter or as a receiver. In addition to transmitter and receiver, the I²C bus uses the concept of master and slave. Table 581 lists the I²C device types.

Table 581. I²C Bus Definitions

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.
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 81341 and 81342 acting as a master on the bus (Figure 140). The 81341 and 81342, as a master, addresses an EEPROM as a slave to receive data. The 81341 and 81342 is a master-transmitter and the EEPROM is a slave-receiver. When the 81341 and 81342 reads data, the 81341 and 81342 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 140. 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.

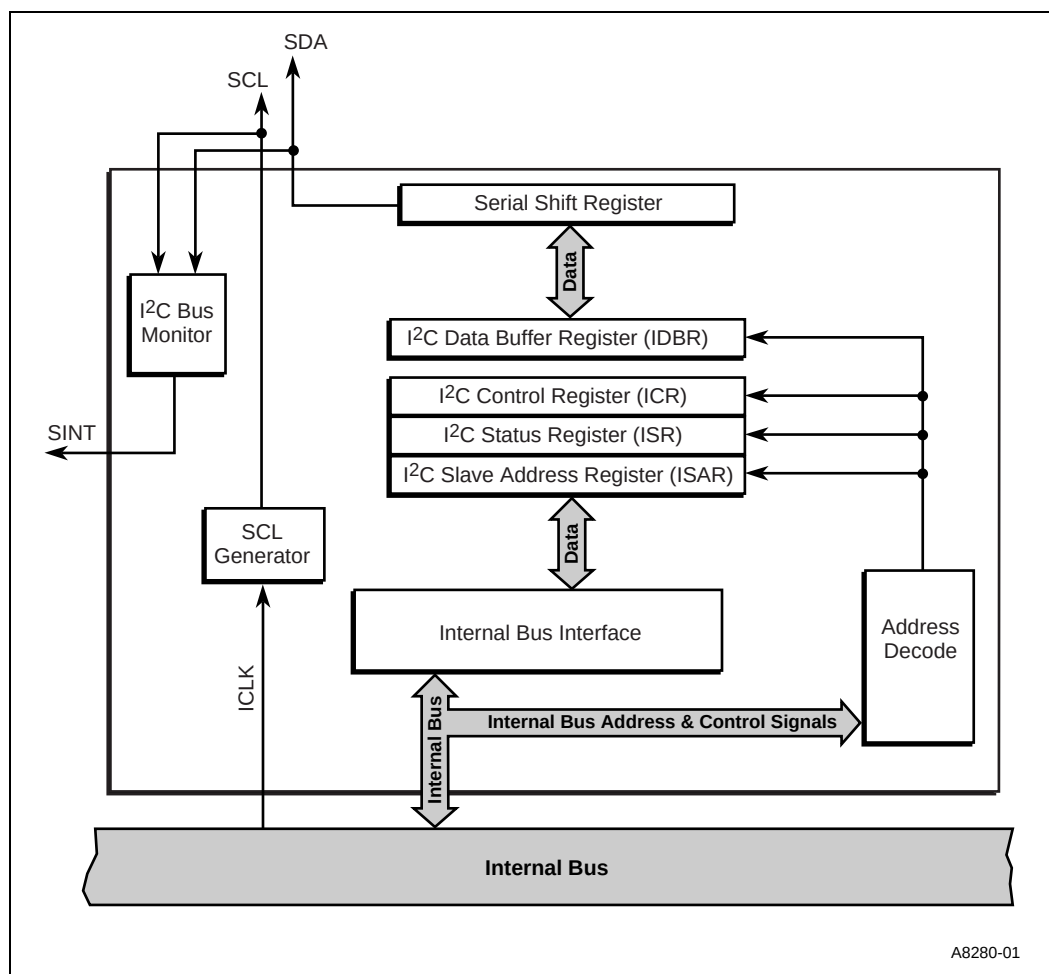
I²C transactions are either initiated by the 81341 and 81342 as a master or are received by the processor as a slave. Both conditions may result in the processor doing reads, writes, or both to the I²C bus.

16.2.1 Operational Blocks

The I²C Bus Interface Unit is a slave peripheral device that is connected to the internal bus. The 81341 and 81342 interrupt mechanism can be used for notifying the 81341 and 81342 that there is activity on the I²C bus. Polling can be also be used instead of interrupts, although it would be very cumbersome. Figure 141 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 81341 and 81342, a set of control and status registers, and a shift register for parallel/serial conversions.

Figure 141. I²C Bus Interface Unit Block Diagram





The I²C interrupts are signalled through a single pin which provides a level sensitive interrupt to the 81341 and 81342 interrupt control unit. The I²C Bus Interface Unit can cause 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 [Section 16.8.2, "I²C Status Register x — ISRx" on page 935](#) 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 81341 and 81342 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 which are at offset (+2140H to +21BFH). The registers and their function are defined in [Section 16.8](#).

The I²C Bus Interface Unit supports fast mode operation of 400 Kbits/sec. Fast mode logic levels, formats, and capacitive loading, and protocols are exactly the same as the 100 Kbits/sec standard mode. Because the data setup and hold times differ between the fast and standard mode, the I²C is designed to meet the slower, standard mode requirements for these two specifications. Refer to the I²C Bus Specification for details.



16.2.2 I²C Bus Interface Modes

The I²C Bus Interface Unit can be in different modes of operation to accomplish a transfer. [Table 582](#) summarizes the different modes.

Table 582. Modes of Operation

Mode	Definition
Master — Transmit	• I²C Bus Interface Unit acts as a master. • Used for a write operation. • I²C Bus Interface Unit sends the data. • I²C Bus Interface Unit is responsible for clocking. • Slave device is in slave-receive mode
Master — Receive	• I²C Bus Interface Unit acts as a master. • Used for a read operation. • I²C Bus Interface Unit receives the data. • I²C Bus Interface Unit is responsible for clocking. • Slave device is in slave-transmit mode
Slave — Transmit	• I²C Bus Interface Unit acts as a slave. • Used for a read (master) operation. • I²C Bus Interface Unit sends the data. • Master device is in master-receive mode.
Slave — Receive (default)	• I²C Bus Interface Unit acts as a slave. • Used for a write (master) operation. • I²C Bus Interface Unit receives the data. • Master device is 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 81341 and 81342.

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 either remains in Slave-Receive mode or transitions 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). When the R/W# bit is low, the master initiating the transaction intends to do a write and the I²C Bus Interface Unit remains in Slave-Receive mode. When 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 [Section 16.3.5, "Slave Operations" on page 925](#).

When the 81341 and 81342 wants to initiate a read or write on the I²C bus, the I²C Bus Interface Unit transitions from the default Slave-Receive mode to Master-Transmit mode. When the 81341 and 81342 wants to write data, the interface remains in Master-Transmit mode after the address transfer has completed. (see [Section 16.2.3.1, "START Condition" on page 916](#) for START information). When the 81341 and 81342 wants to read data, the I²C Bus Interface Unit transmits the start address, then transition to Master-Receive mode. Master operation is further defined in [Section 16.3.4, "Master Operations" on page 922](#).



16.2.3 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 81341 and 81342 uses 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

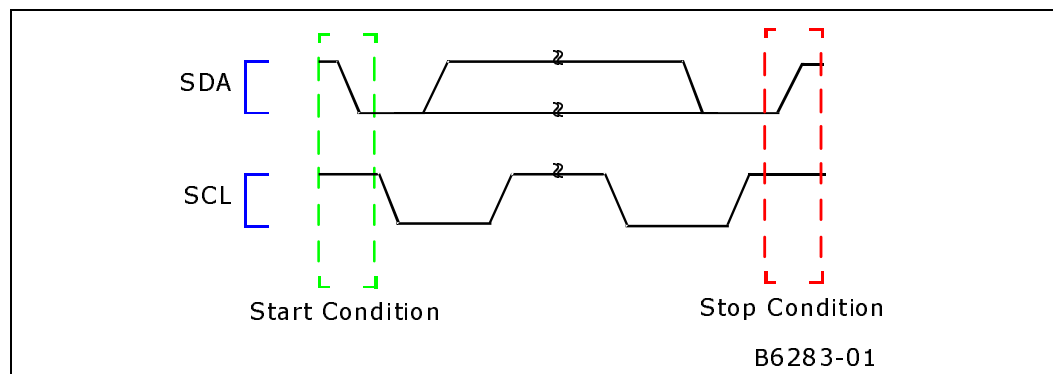
Table 583 summarizes the definition of the START and STOP bits in the ICR.

Table 583. 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 I²C Bus Interface Unit. This is used when multiple data bytes need to be transferred.
0	1	START Condition and Repeated START	• The I²C Bus Interface Unit sends 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 contains the target slave address and the R/W# bit. This enables multiple transfers to different slaves without giving up the bus. • The interface stays in Master-Transmit mode when a write is used or transition to master-receive mode when a read is requested.
1	X	STOP Condition	• In Master-Transmit mode, the I²C Bus Interface Unit transmits the 8-bit IDBR and then send a STOP on the I²C bus. • In Master-Receive mode, the Ack/Nack Control bit in the ICR must be changed to a negative Ack (see Section 16.3.2). The I²C Bus Interface Unit writes the Nack bit (Ack/Nack Control bit must be 1), receive the data byte in the IDBR, then send a STOP on the I²C bus.

Figure 142 shows the relationship between the **SDA** and **SCL** lines for a START and STOP condition.

Figure 142. Start and Stop Conditions



16.2.3.1 START Condition

The START condition (bits 1:0 of the ICR set to 01₂) initiates a master transaction or repeated START. Software must load the target slave address and the R/W# bit in the IDBR (see [Section 16.8.4, “I²C Data Buffer Register x — IDBRx” on page 938](#)) 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 [Section 16.3.3, “Arbitration” on page 920](#) for details on how the I²C unit functions under those circumstances.

16.2.3.2 No START or STOP Condition

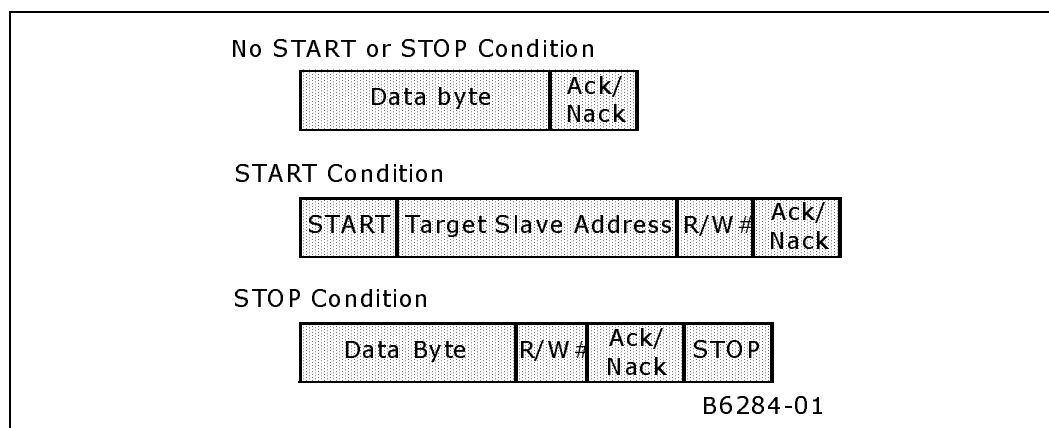
No START or STOP condition (bits 1:0 of the ICR set to 00₂) is used in master-transmit mode while the 81341 and 81342 is transmitting multiple data bytes (see [Figure 142](#)). Software writes the data byte, sets the IDBR Transmit Empty bit in the ISR (and interrupt when enabled), and clears the Transfer 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.

16.2.3.3 STOP Condition

The STOP condition (bits 1:0 of the ICR set to 10₂) 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 142](#)). In master-receive mode, to initiate the last transfer the 81341 and 81342 must set the Ack/Nack bit, the STOP bit, and the Transfer Byte bit in the ICR. Software must clear the STOP condition after it is transmitted.

Figure 143. START and STOP Conditions





16.3 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.
9. Serial Clock Line (**SCL**) Generation.

The 81341 and 81342's I²C unit 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 Fast Mode Enable bit, which is programmed at initialization. The following equation is used to determine the **SCL** transition period:

Equation 35. SCL Transition Period

$\text{SCL Transition Period} = (30 \text{ ns}) * (167 - (\text{Fast Mode Enable} * 120))$

16.3.1 Data and Addressing Management

Data and slave addressing is managed via the I²C Data Buffer Register (IDBR) and I²C Slave Address Register (ISAR). IDBR (see [Section 16.8.4, "I²C Data Buffer Register x — IDBRx"](#)) contains data or a slave address and R/W# bit. ISAR contains the 81341 and 81342 programmable slave address. Data coming into the I²C unit is received into IDBR after a full byte is received and acknowledged. To transmit data, the processor writes to IDBR, and the I²C unit passes this onto the serial bus when the Transfer Byte bit in the ICR is set. See [Section 16.8.1, "I²C Control Register x — ICRx"](#).

When the I²C unit is in transmit mode (master or slave):

1. Software writes data to the IDBR over the internal bus. This initiates a master transaction or sends the next data byte, after the IDBR Transmit Empty bit is sent.
2. I²C unit transmits data from IDBR when the Transmit Empty bit in the ICR 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 processor has written the IDBR (and a STOP condition is not in place), the I²C unit inserts wait states until the processor writes a new value into the IDBR and sets the ICR Transfer Byte bit.

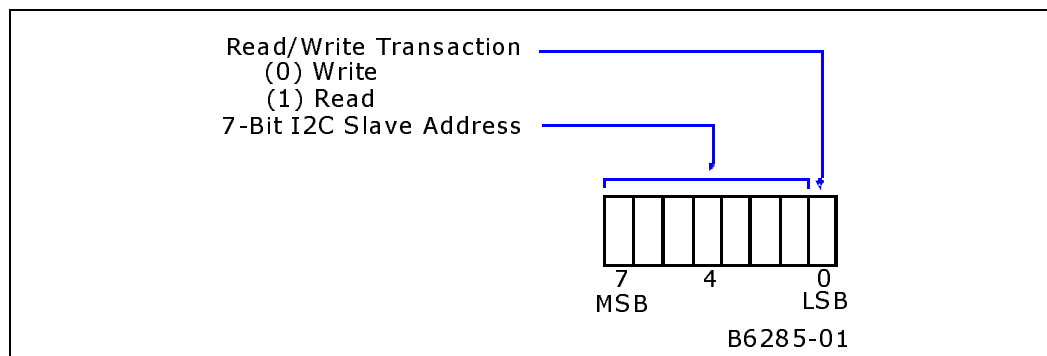
When the I²C unit is in receive mode (master or slave):

1. The processor reads the IDBR data over the internal bus after the IDBR Receive Full interrupt is signalled.
2. I²C unit transfers data from shift register to IDBR after the Ack cycle completes.
3. The I²C unit inserts wait states until the IDBR is read. Refer to [Section 16.3.2, "I²C Acknowledge" on page 919](#) for acknowledge pulse information in receiver mode.
4. After processor reads IDBR, the I²C unit writes the ICRs Ack/Nack Control bit and the Transfer Byte bit, allowing the next byte transfer to proceed.

16.3.1.1 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 Figure 144).

Figure 144. 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). See [Section 16.3.6, "General Call Address" on page 927](#) for actions when a general call address is detected.



16.3.2 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 145).

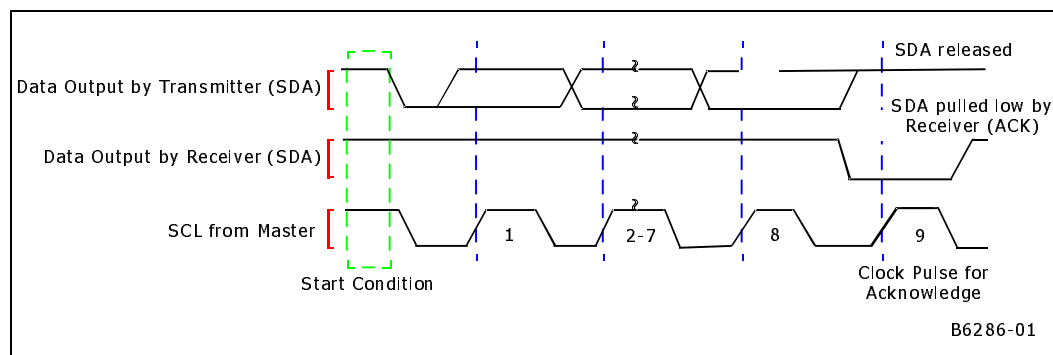
In master-transmit mode, when the target slave receiver device cannot generate the acknowledge pulse, the **SDA** line remains high. This lack of acknowledge (Nack) causes the I²C unit to set the bus error detected bit in the ISR and generate the associated interrupt (when enabled). The I²C unit aborts 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) remains set until a STOP or repeated START is received.

Figure 145. Acknowledge on the I²C Bus



16.3.3 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.

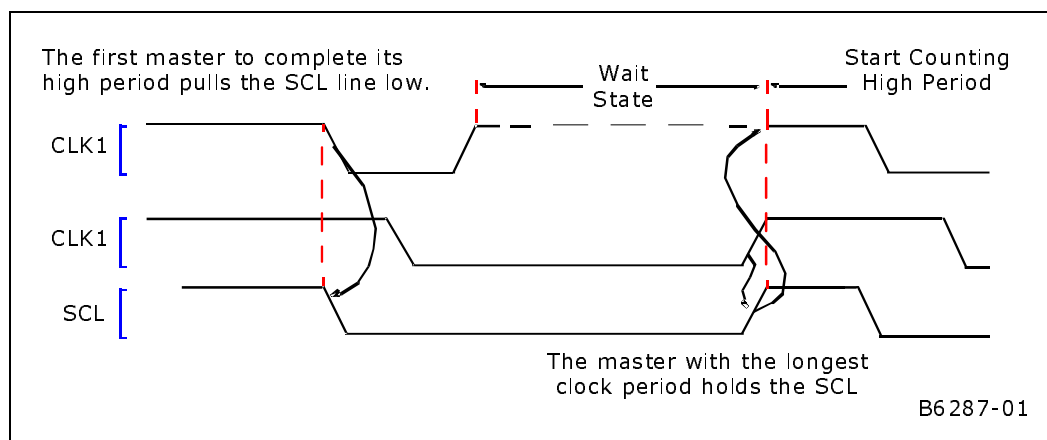
Arbitration can continue for a long period. When the address bit and the R/W# are the same, the arbitration moves to the data. Due to the wired-AND nature of the I²C bus, no data is lost when both (or all) masters are outputting the same bus states. When the address, the R/W# bit, or the data are different, the master which outputted the high state (master's data is different from **SDA**) loses arbitration and shut its data drivers off. When losing arbitration, the I²C Bus Interface Unit shuts 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.

16.3.3.1 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 146). 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 146. Clock Synchronization During the Arbitration Procedure

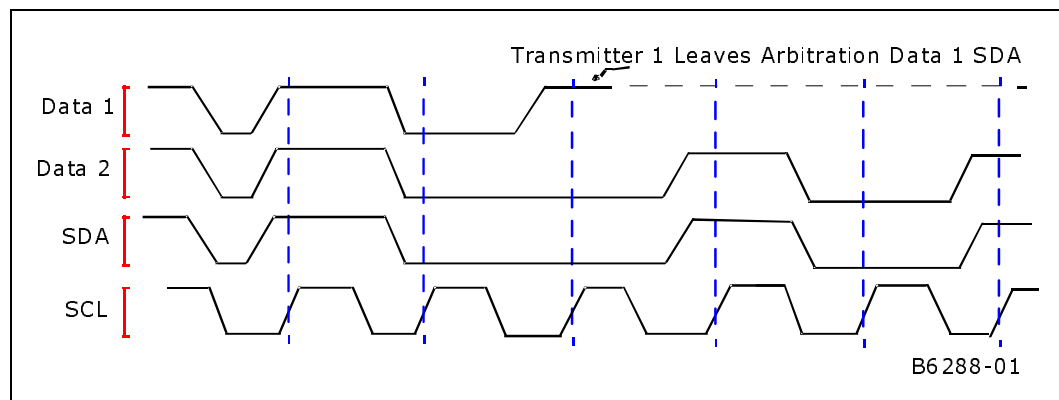




16.3.3.2 SDA Arbitration

Arbitration on the **SDA** line can continue for a long period, starting with address and R/W# bits and continuing with data bits. Figure 147 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 and R/W# are the same, arbitration moves to the data. 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 address, R/W#, or data is different, the master that output the first low 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 remainder of byte transfer, sets arbitration loss detected ISR bit, then returns to idle (Slave-Receive) mode.

Figure 147. Arbitration Procedure of Two Masters



When the I²C unit loses arbitration during transmission of the seven address bits and the 81341 and 81342 is not being addressed as a slave device, the I²C unit re-sends the address when the I²C bus becomes free. This is possible because the IDBR and ICR registers are not overwritten when arbitration is lost.

When the arbitration loss is due to another bus master addressing the 81341 and 81342 as a slave device, the I²C unit switches to slave-receive mode and the original data in the I²C data buffer register is overwritten. Software is responsible for clearing the start and re-initiating the master transaction at a later time.

Note: Software must not allow the I²C unit to write to its own slave address. This can cause the I²C bus to enter 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 (slave can discard master address).



16.3.4 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 81341 and 81342 writes data
- Master-Receive — The 81341 and 81342 reads data

The 81341 and 81342 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 584 describes the I²C Bus Interface Unit responsibilities as a master device.

Table 584. 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 SCL Enable bit must be set. • The Unit Enable bit must be set.
Write target slave address to IDBR	Master-transmit Master-receive	• The Intel XScale® processor writes to IDBR bits 7-1 before a START condition is enabled. • First 7 bits sent on bus after START. • See Section 16.2.3.
Write R/W# Bit to IDBR	Master-transmit Master-receive	• The Intel XScale® processor writes to the least significant IDBR bit with the target slave address. • When low, the master remains a master-transmitter. When high, the master transitions to a master-receiver. • See Section 16.3.1.
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. • Intel XScale® processor sets the START bit. • Intel XScale® processor sets the Transfer Byte bit which initiates the start condition. • See Section 16.2.3.
Initiate first data byte transfer	Master-transmit Master-receive	• Intel XScale® processor 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	• When two or more masters signal a start within the same clock period, arbitration must occur. • The I²C Bus Interface Unit arbitrates 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. • When the I²C Bus Interface Unit loses arbitration, it sets the Arbitration Loss Detect ISR bit after byte transfer is complete and transition to slave-receive (default) mode. • When I²C Bus Interface Unit loses arbitration while attempting to send the target address byte, the I²C Bus Interface Unit attempts to resend it when the bus becomes free. • The system designer must ensure the boundary conditions described in Section 16.3 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. When enabled, the IDBR Transmit Empty Interrupt is signalled to the Intel XScale® processor. • Intel XScale® processor writes 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 when requested.

**Table 584. Master Transactions (Sheet 2 of 2)**

I ² C Master Action	Mode of Operation	Definition
Wait for Acknowledge from slave-receiver	Master-transmit only	- As a master-transmitter, the I²C Bus Interface Unit generates 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 Section 16.3.2.
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 Intel XScale® processor reads the IDBR when the IDBR Receive Full bit is set and the Transfer Byte bit is clear. When enabled, a IDBR Receive Full Interrupt is signalled to the Intel XScale® microarchitecture. - When the IDBR is read, when the Ack/Nack Status is clear (indicating Ack), the Intel XScale® processor writes the Ack/Nack Control bit and set the Transfer Byte bit to initiate the next byte read. - When 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. - When the Ack/Nack Status bit is set (indicating Nack), Transfer Byte bit is clear, but the STOP bit is clear, then the Intel XScale® processor has two options: 1. set the START bit, write a new target address to the IDBR, and set the Transfer Byte bit which sends a repeated start condition, 2. set the Master Abort bit and leave the Transfer Byte clear which sends a STOP only.
Transmit Acknowledge to slave-transmitter	Master-receive only	- As a master-receiver, the I²C Bus Interface Unit generates the clock for the acknowledge pulse. The I²C Bus Interface Unit is also responsible for driving the SDA line during the Ack cycle. - When the next data byte is to be the last transaction, the Intel XScale® processor sets the Ack/Nack Control bit for Nack generation. - See Section 16.3.2.
Generate a Repeated START to chain I ² C transactions	Master-transmit Master-receive	- When 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 Intel XScale® processor writes the next target slave address and the R/W# bit to the IDBR, set the START bit, and set the Transfer Byte bit. - See Section 16.2.3.
Generate a STOP	Master-transmit Master-receive	- Generated after the Intel XScale® processor writes the last data byte on the bus. - Intel XScale® processor generates a STOP condition by setting the STOP bit in the ICR. - See Section 16.2.3.

When the 81341 and 81342 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 148). 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 149). Figure 150 shows the wave forms of **SDA** and **SCL** for a complete data transfer.

Figure 148. Master-Receiver Read from Slave-Transmitter

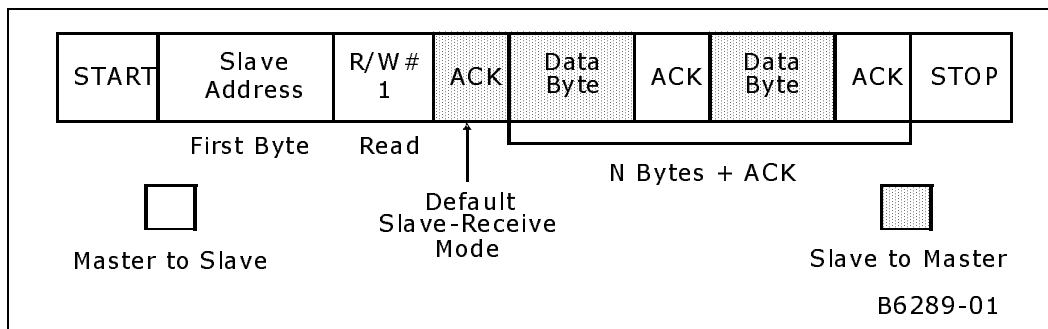


Figure 149. Master-Receiver Read from Slave-Transmitter / Repeated Start / Master-Transmitter Write to Slave-Receiver

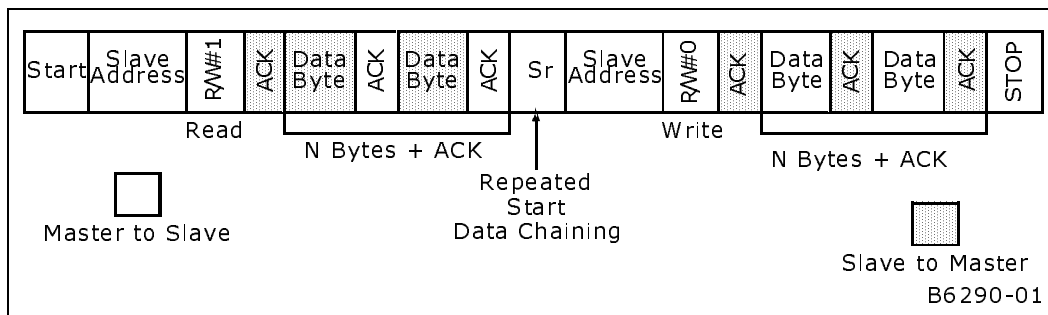
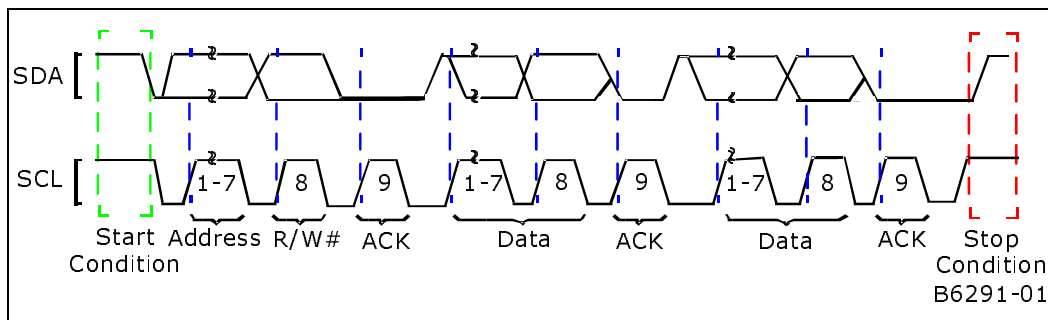


Figure 150. A Complete Data Transfer





16.3.5 Slave Operations

Table 585 describes the I²C Bus Interface Unit's responsibilities as a slave device.

Table 585. 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). When there is a match, the I²C Bus Interface Unit sends an Ack. - When the first 8 bits are all zeros, this is a general call address. When the General Call Disable bit is clear, both the General Call Address Detected bit and the Slave Mode Operation bit in the ISR are set. See Section 16.3.6. - When 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 Mode Operation bit is cleared. When the R/W# bit is high, the I²C Bus Interface Unit transitions to slave-transmit mode and the Slave Mode Operation bit is set.
Setting the Slave Address Detected bit	Slave-receive Slave-transmit	- Indicates the interface has detected an I²C operation that addresses the 81341 and 81342 (this includes general call address). The Intel XScale® processor can distinguish an ISAR match from a General Call by reading the General Call Address Detected bit. - An interrupt is signalled (when 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. When enabled, the IDBR Receive Full Interrupt is signalled to the Intel XScale® processor. - Intel XScale® processor reads 1 data byte from the IDBR. When the IDBR is read, the Intel XScale® processor writes 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 Section 16.3.2.
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. When enabled, the IDBR Transmit Empty Interrupt is signalled to the Intel XScale® processor. - Intel XScale® processor writes 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 Section 16.3.2.

Figure 151 through Figure 153 are examples of I²C transactions. These show the relationships between master and slave devices.

Figure 151. Master-Transmitter Write to Slave-Receiver

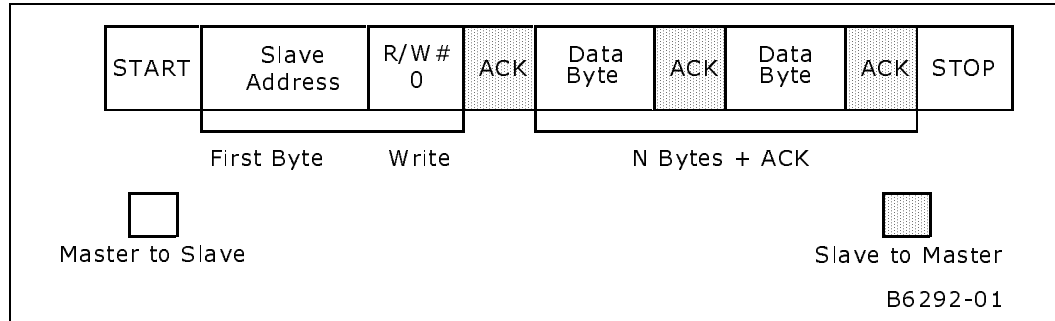


Figure 152. Master-Receiver Read to Slave-Transmitter

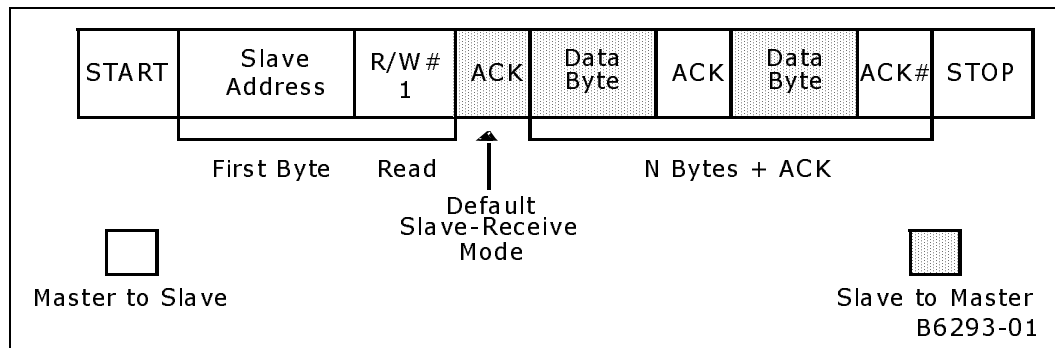
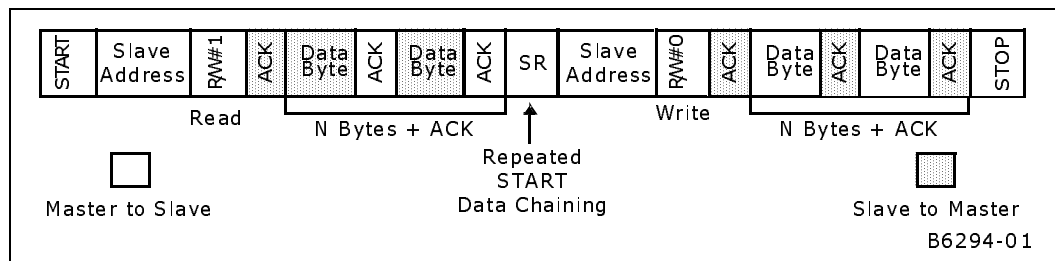


Figure 153. Master-Receiver Read to Slave-Transmitter, Repeated START, Master-Transmitter Write to Slave-Receiver





16.3.6 General Call Address

The I²C unit supports both sending and receiving general call address transfers on the I²C bus. When sending a general call message from the I²C unit, software must set the General Call Disable bit in the ICR to keep the I²C unit from responding as a slave. Failure to set this bit causes the I²C Bus to enter an indeterminate state.

A general call address is defined as 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 can ignore the general call address. The second and following bytes of a general call transaction are acknowledged by every device using it on the bus. Any device not using these bytes must not Ack. The meaning of a general call address is defined in the second byte sent by the master-transmitter. Figure 154 shows a general call address transaction. The least significant bit (B) of the second byte defines the transaction. Table 586, "General Call Address Second Byte Definitions" on page 927 shows the valid values and definitions when B=0.

When the 81341 and 81342 is acting as a slave, and the I²C unit receives a general call address and the ICR General Call Disable bit is clear the I²C unit:

- Sets the ISR general call address detected bit
- Sets the ISR slave address detected bit
- Interrupts (when enabled) the 81341 and 81342

When 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 154. General Call Address

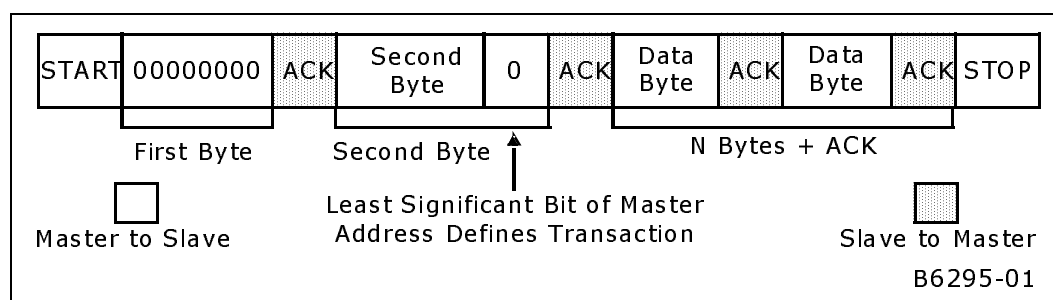


Table 586. General Call Address Second Byte Definitions

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

When directed to reset, the I²C Bus Interface Unit returns to its default reset condition with the exception of the ISAR. The 81341 and 81342 is responsible for ensuring this occurs, not the I²C Bus Interface Unit hardware.

When B=1, the sequence is used as a hardware general call by hardware masters only they cannot transmit a slave address, only their own address. The I²C Bus Interface Unit does not support this mode of operation.

I²C 10-bit addressing and CBUS compatibility are not supported.



16.4 Slave Mode Programming Examples

16.4.1 Initialize Unit

1. Write ISAR: Set slave address
2. Write ICR: Enable all interrupts, set Unit Enable

16.4.2 Write 1 Byte as a Slave

1. Wait for Slave Address Detected interrupt.
Read ISR: Slave Address Detected (set), Unit Busy (set), R/W# bit (1), Ack/Nack (Clear - Ack)
2. Write IDBR: Load data byte to transfer
3. Write ICR: Set Transfer Byte bit
4. Wait for IDBR Transmit Empty interrupt.
Read ISR: IDBR Transmit Empty (set), Ack/Nack (set - indicates last byte write), R/W# bit (0)
5. Clear interrupt by clearing the IDBR Transmit Empty Interrupt bit.
6. Wait for interrupt.
Read ISR: Unit Busy (clear), Slave STOP Detected (set)
7. Clear interrupt by clearing Slave STOP Detected Interrupt bit.

16.4.3 Read 2 Bytes as a Slave

1. Wait for Slave Address Detected interrupt.
Read ISR: Slave Address Detected (set), Unit busy (set), R/W# bit (0)
2. Read byte 1 on I²C bus
Write ICR: Set Transfer Byte bit to initiate the transfer
3. Wait for interrupt.
Read ISR: IDBR Receive Full (set), Ack/Nack (clear), R/W# bit (0)
Clear interrupt by clearing IDBR Receive Full bit.
Read IDBR: To get the data.
4. Read byte 2 on I²C bus
Write ICR: Set Transfer Byte bit to initiate the transfer
5. Wait for interrupt.
Read ISR: IDBR Receive Full (set), Ack/Nack (clear), R/W# bit (0)
Clear interrupt by clearing IDBR Receive Full bit.
Read IDBR: To get the data.
Write ICR: Set Transfer Byte bit (to release I²C bus allowing next transfer)
6. Wait for interrupt.
Read ISR: Unit busy (clear), Slave STOP Detected (set)
Clear interrupt by clearing Slave STOP Detected bit.



16.5 Master Programming Examples

16.5.1 Initialize Unit

1. Write ISAR: Set slave address
2. Write ICR: Enable all interrupts (except Arb Loss), set **SCL** Enable, set Unit Enable

16.5.2 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, Set Transfer Byte bit to initiate the access
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
Read status register: IDBR Transmit Empty (set), Unit Busy (set), R/W# bit (clear)
Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.

Note: Arbitration Loss Detected bit may be set. When arbitration was lost, because Arb Loss interrupt was disabled, an address retry occurs when bus becomes free. Clear Arbitration Loss Detected bit when set.

4. Send byte with STOP
Write IDBR: With data byte to send
Write ICR: Clear START bit, Set STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit to initiate the access
5. Wait for Buffer empty interrupt. When interrupt arrives (Note: Unit is sending STOP):
Read status register: IDBR Transmit Empty (set), Unit busy (set - maybe), R/W# bit (clear)
Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.
Clear ICR STOP bit (optional)
Wait until Unit busy is clear before clearing the ICR SCL Enable bit.

16.5.3 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 status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (set)
Clear IDBR Transmit Empty bit to clear the interrupt.
4. 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
5. Wait for Buffer full interrupt. When interrupt arrives (Note: Unit is sending STOP):
Read status register: IDBR Receive Full (set), Unit Busy (set - maybe), R/W# bit (Set), Ack/Nack bit (Set)
Clear IDBR Receive Full bit to clear the interrupt.
Read IDBR data.
Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional)
Wait until Unit busy is clear before clearing the ICR SCL Enable bit. (optional)



16.5.4 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 (set), Unit busy (set), R/W# bit (clear)
Clear IDBR Transmit Empty bit to clear the interrupt.
4. Send byte 1
Write IDBR: With data byte to send
Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit to initiate the access
5. Wait for Buffer empty interrupt.
Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (clear)
Clear IDBR Transmit Empty bit to clear the interrupt.
6. Send byte 2
Write IDBR: With data byte to send
Write ICR: Clear START bit, Clear STOP bit, Set Transfer Byte bit to initiate the access
7. Wait for Buffer empty interrupt.
Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (clear)
Clear IDBR Transmit Empty bit to clear the interrupt.
8. Send repeated start as a master
Write IDBR: Target slave address and R/W# bit (1 for read)
Write ICR: Set START bit, Clear STOP bit, Disable Arb Loss interrupt, Set Transfer Byte bit to initiate the access
9. Wait for IDBR Transmit Empty interrupt. When interrupt comes.
Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (set)
Clear IDBR Transmit Empty bit to clear the interrupt.
10. 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
11. Wait for Buffer full interrupt. When interrupt comes (Note: Unit is sending STOP).
Read status register: IDBR Receive Full (set), Unit busy (set - maybe), R/W# bit (Set), Ack/Nack bit (Set)
Clear IDBR Receive Full bit to clear the interrupt.
Read IDBR data.
Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional)
Wait until Unit busy is clear before clearing the ICR SCL Enable bit. (optional)



16.5.5 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 (set), Unit Busy (set), R/W# bit (set)
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 (set), Unit busy (set), R/W# bit (Set), Ack/Nack bit (Clear)
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 are 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 (set), Unit Busy (set), R/W# bit (Set), Ack/Nack bit (Set)
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.



16.6 Glitch Suppression Logic

The I²C Bus Interface Unit has built-in glitch suppression logic. Glitches are suppressed according to: $2 * I^2C \text{ clock period}$. For example, with the 33 MHz (30. ns period) I²C clock glitches of 60ns or less are suppressed. This is within the 50 ns glitch suppression specified.

16.7 Reset Conditions

The I²C unit is reset with internal bus reset. 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 with the exception of the ISAR. ISAR is not affected by a reset.

When the Unit Reset bit in the ICRx is set, only the 81341 and 81342 I²C unit resets, the associated I²C MMRs remain intact. When resetting the I²C unit with the ICRx unit reset, use the following guidelines:

1. In the ICRx register, set the reset bit and clear the remainder of the register.
2. Clear the ISRx register.
3. Clear reset in the ICRx.

16.8 Register Definitions

The following registers are associated with the I²C Bus Interface Units. Each I²C Bus Interface Unit has five memory-mapped control registers for independent operation. In register titles, x is 0 or 1 for unit 0 or 1, respectively.

They are all located within the peripheral memory- mapped address space of the 81341 and 81342.

Table 587. I²C Register Summary

Section, Register Name, Acronym, Page
Section 16.8.1, "I ² C Control Register x — ICRx" on page 933
Section 16.8.2, "I ² C Status Register x — ISRx" on page 935
Section 16.8.3, "I ² C Slave Address Register x — ISARx" on page 937
Section 16.8.4, "I ² C Data Buffer Register x — IDBRx" on page 938
Section 16.8.5, "I ² C Bus Monitor Register x — IBMRx" on page 939
Section 16.8.6, "I ² C Manual Bus Control Register x — IMBCRx" on page 940

16.8.1 I²C Control Register x — ICRx

The 81341 and 81342 uses the bits in the I²C Control Register (ICRx) to control the I²C unit.

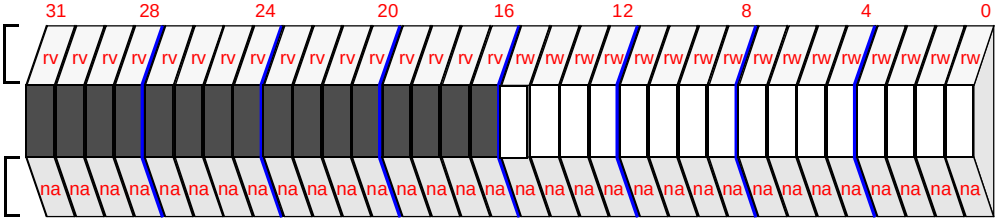
Table 588. I²C Control Register x — ICRx (Sheet 1 of 2)

Unit #	Intel XScale® processor internal bus address offset	Attribute Legend:	RW = Read/Write
0	+2500H	RV = Reserved	RC = Read Clear
1	+2520H	PR = Preserved	RO = Read Only
2	+2540H	RS = Read/Set	NA = Not Accessible

Bit	Default	Description
31:16	0000H	Reserved
15	0	Fast Mode: 0 = 100 KBit/sec operation 1 = 400 KBit/sec operation
14	0 ₂	Unit Reset: 0 = No reset. 1 = Reset the I ² C unit only.
13	0 ₂	Slave Address Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables I ² C unit to interrupt the 81341 and 81342 upon detecting a slave address match or a general call address.
12	0 ₂	Arbitration Loss Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the 81341 and 81342 upon losing arbitration while in master mode.
11	0 ₂	Slave STOP Detected Interrupt Enable: 0 = Disable interrupt. 1 = Enables I ² C unit to interrupt the 81341 and 81342 when it detects a STOP condition in slave mode.
10	0 ₂	Bus Error Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the 81341 and 81342 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 insuring that misplaced START and STOP conditions do not occur. See Section 16.6, "Glitch Suppression Logic" on page 932.
09	0 ₂	IDBR Receive Full Interrupt Enable: 0 = Disable interrupt. 1 = Enables I ² C unit to interrupt the 81341 and 81342 when IDBR has received a data byte from the I ² C bus.
08	0 ₂	IDBR Transmit Empty Interrupt Enable: 0 = Disable interrupt. 1 = Enables the I ² C unit to interrupt the 81341 and 81342 after transmitting a byte onto the I ² C bus.
07	0 ₂	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.
06	0 ₂	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 insure the I ² C bus is idle before setting this bit.



Table 588. I²C Control Register x — ICRx (Sheet 2 of 2)

																																	
Unit #	Intel XScale® processor internal bus address	Attribute Legend:										RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																					
0	offset																																
1	+2500H																																
2	+2520H +2540H																																
Bit	Default	Description																															
05	0 ₂	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.																															
04	0 ₂	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 clear 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 81341 and 81342 does not send a repeated START, setting this bit sends the STOP. Once again, the Transfer Byte bit (03) must remain clear.																															
03	0 ₂	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 81341 and 81342 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.																															
02	0 ₂	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.																															
01	0 ₂	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 16.2.3.3, "STOP Condition" on page 916 for more details on the STOP state. 0 = Do not send a STOP. 1 = Send a STOP.																															
00	0 ₂	START: used to initiate a START condition to the I ² C unit when in master mode. See Section 16.2.3.1, "START Condition" on page 916 for more details on the START state. 0 = Do not send a START. 1 = Send a START.																															



16.8.2 I²C Status Register x — ISR_x

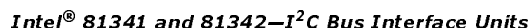
I²C interrupts are signalled to the 81341 and 81342 interrupt controller by the I²C Interrupt Status Register (ISR_x). Software uses the ISR bits to check the status of the I²C unit and bus. ISR_x bits (bits 9-5) are updated after the Ack/Nack bit has completed on the I²C bus.

The ISR_x is also used to clear interrupts signalled from the I²C Bus Interface Unit. These are:

- IDBR_x Receive Full
- IDBR_x Transmit Empty
- Slave Address Detected
- Bus Error Detected
- STOP Condition Detect
- Arbitration Lost

Table 589. I²C Status Register x — ISR_x (Sheet 1 of 2)

Unit #	Intel XScale® processor internal bus address	Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible	
0	offset		
1	+2504H		
2	+2524H		
	+2544H		
Bit	Default	Description	
31:11	000000H	Reserved	
10	0 ₂	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 insure that misplaced START and STOP conditions do not occur. See Section 16.3.3, "Arbitration" on page 920 .	
09	0 ₂	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.	
08	0 ₂	General Call Address Detected: 0 = No general call address received. 1 = I ² C unit received a general call address.	
07	0 ₂	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.	
06	0 ₂	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.	



Unit # Intel XScale® processor internal bus address offset

0 +2504H

1 +2524H

2 +2544H

Attribute Legend:

RV = Reserved

PR = Preserved

RS = Read/Set

RW = Read/Write

RC = Read Clear

RO = Read Only

NA = Not Accessible

Bit	Default	Description
05	0 ₂	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.
04	0 ₂	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.
03	0 ₂	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 81341 and 81342 I ² C unit is not involved in the transaction.
02	0 ₂	Unit Busy: 0 = I ² C unit not busy. 1 = Set when the 81341 and 81342 I ² C unit is busy. This is defined as the time between the first START and STOP.
01	0 ₂	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.
00	0 ₂	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.



16.8.3 I²C Slave Address Register x — ISARx

The I²C Slave Address Register (ISARx) (see Table 590) defines the I²C unit 7-bit slave address to which the 81341 and 81342 responds when in slave-receive mode. This register is written by the 81341 and 81342 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 is not affected by the 81341 and 81342 being reset. The ISAR register default value is 0000000₂.

Table 590. I²C Slave Address Register x — ISARx

Unit #	Intel XScale® processor internal bus address	Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
0	offset																												
1	+2508H																												
2	+2528H +2548H																												
Bit	Default	Description																											
31:07	000000H	Reserved																											
06:00	00H	I ² C Slave Address : The 7-bit address to which the I ² C unit responds when in slave-receive mode.																											



16.8.4 I²C Data Buffer Register x — IDBRx

The I²C Data Buffer Register (IDBRx) is used by the 8134881341 and 81342 to transmit and receive data from the I²C bus. The accesses the IDBRx by the 8134881341 and 81342 on one side and by the I²C shift register on the other. Data coming into the I²C Bus Interface Unit is received into the IDBRx after a full byte has been received and acknowledged. Data going out of the I²C Bus Interface Unit is written to the IDBRx by the Intel XScale® processor and sent to the serial bus.

When the I²C Bus Interface Unit is in transmit mode (master or slave), the 8134881341 and 81342 writes data to the IDBRx over the internal bus. This occurs when a master transaction is initiated or when the IDBRx Transmit Empty Interrupt is signalled. Data is moved from the IDBRx to the shift register when the Transfer Byte bit is set. The IDBR Transmit Empty Interrupt is signalled (when enabled) when a byte has been transferred on the I²C bus and the acknowledge cycle is complete. When the IDBRx is not written by the 8134881341 and 81342 (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 inserts wait states until the Intel XScale® processor writes the IDBRx and sets the Transfer Byte bit.

When the I²C Bus Interface Unit is in receive mode (master or slave), the processor reads IDBRx data over the internal bus. This occurs when the IDBRx Receive Full Interrupt is signalled. The data is moved from the shift register to the IDBRx when the Ack cycle is complete. The I²C Bus Interface Unit inserts wait states until the IDBR has been read. Refer to [Section 16.3.2, "I²C Acknowledge" on page 919](#) for acknowledge pulse information in receiver mode. After the 8134881341 and 81342 reads the IDBRx, 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 IDBRx register is 00H after reset.

Table 591. I²C Data Buffer Register x — IDBRx

Unit #	Intel XScale® processor internal bus address	Attribute Legend:
0	offset	RW = Read/Write
1	+250CH	RV = Reserved
2	+252CH	PR = Preserved
	+254CH	RS = Read/Set
		RC = Read Clear
		RO = Read Only
		NA = Not Accessible
Bit	Default	Description
31:08	000000H	Reserved
07:00	00H	I ² C Data Buffer: Buffer for I ² C bus send/receive data.



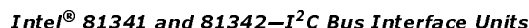
16.8.5 I²C Bus Monitor Register x — IBMRx

The I²C Bus Monitor Register (IBMRx) 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.

Table 592. I²C Bus Monitor Register x — IBMRx

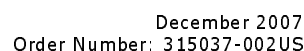
Unit #	Intel XScale® processor internal bus address	Attribute Legend:
0	offset	RW = Read/Write
1	+2514H	RV = Reserved
2	+2534H	RC = Read Clear
	+2554H	PR = Preserved
		RO = Read Only
		RS = Read/Set
		NA = Not Accessible

Bit	Default	Description
31:02	0	Reserved
01	1	SCL Status: This bit continuously reflects the value of the SCL pin.
00	1	SDA Status: This bit continuously reflects the value of the SDA pin.



The I²C Manual Bus Control Register (IMBCRx) can be used to manually release or pull down the **SCL** and **SDA** pins. The values of these pins are controlled by the IMBCRx bits 2:1 when bit 0 of the IMBCRx is set. When software determines that the I²C bus is hung using the “[I2C Bus Monitor Register x — IBMRx” on page 939](#), this register may be used to force the I²C bus out of the hung state.

Table 593. I²C Manual Bus Control Register x — IMBCRx





17.0 General Purpose I/O Unit

This chapter describes the Intel® 81341 and 81342 I/O Processors (81341 and 81342) General Purpose I/O Unit; operation modes, setup, external memory interface, and implementation of General Purpose I/Os (GPIOs).

17.1 General Purpose Input Output Support

Twelve pins are provided as General Purpose Input Output (GPIO) pins. The Twelve pins are **GPIO[15:0]**. These pins can be used by the Intel XScale® processors to control or monitor external devices in the I/O subsystem.

17.1.1 General Purpose Inputs

The current state of the twelve GPIO pins can be read in **HPI#**.

Note: When configured as GPIOs, the twelve GPIO pins can be used as (up to) 12 additional external interrupt inputs dedicated to the Intel XScale® processor. This feature is available on a per pin basis simply by programming the INTCTL[3:0] registers.

17.1.2 General Purpose Outputs

The output function of the GPIO pins is controlled by two registers, as stated in [Section 17.2.3, "GPIO Output Data Register — GPOD" on page 946](#) and [Section 17.2.1, "GPIO Output Enable Register — GPOE" on page 943](#).

The output enables are mapped on a per bit basis to each of the data bits in the GPIO Output Data Register. When a bit of the GPIO Output Enable Register is cleared, the corresponding data bit value in the GPIO Output Data Register is actively driven on the appropriate GPIO pin.

17.1.3 Reset Initialization of General Purpose I/O Function

- GPIO Input Data Register is initialized to the state of **GPIO[15:0]** bus upon assertion of **P_RST#**. Note that **GPIO[3:0]** pins are multiplexed with the PCI-X interrupts **P_INT[D:A]#** and, therefore, are initialized as output pins when the PCI-X interface is used as an endpoint.
- GPIO Output Data Register is initialized to all zeros upon assertion of **P_RST#**.
- GPIO Output Enable Register is initialized to FFFFH upon assertion of **P_RST#**. This means that **GPIO[15:0]** are initialize as inputs.
- **GPIO[15:0]** pins are tristated during **P_RST#** assertion.



17.2 Register Definitions

All GPIO are visible as 81341 and 81342 memory mapped registers and can be accessed through the internal memory bus. Each is a 32-bit register and is memory-mapped in the Intel XScale® processor memory space.

The programmer interface to the General Purpose I/O is through memory-mapped control registers. [Table 594](#) describes these registers.

Table 594. General Purpose I/O Registers Addresses

Register Name	Description	MMR Address Offset
GPOE	GPIO Output Enable Register	2480H
GPID	GPIO Input Data Register	2484H
GPOD	GPIO Output Data Register	2488H



17.2.1 GPIO Output Enable Register — GPOE

The GPIO Output Enable Register, on a per pin basis, enables the output value contained in the GPIO Output Data Register, onto the appropriate pin. The GPIO Output Enable Register is initialized to FFFFH such that all of **GPIO[15:0]** are inputs. In order to enable a particular GPIO pin to operate as an output following the deassertion of **P_RST#**, the user needs to write a 0 into the appropriate GPOE bit.

Table 595. GPIO Output Enable Register — GPOE

Bit	Default	Description
31:16	0000H	Reserved.
15	1 ₂	GPIO15 Output Enable — When clear, bit[15] of the GPIO Output Data Register is enabled onto the GPIO[15] pin.
14	1 ₂	GPIO14 Output Enable — When clear, bit[14] of the GPIO Output Data Register is enabled onto the GPIO[14] pin.
13	1 ₂	GPIO13 Output Enable — When clear, bit[13] of the GPIO Output Data Register is enabled onto the GPIO[13] pin.
12	1 ₂	GPIO12 Output Enable — When clear, bit[12] of the GPIO Output Data Register is enabled onto the GPIO[12] pin.
11	1 ₂	GPIO11 Output Enable — When clear, bit[11] of the GPIO Output Data Register is enabled onto the GPIO[11] pin.
10	1 ₂	GPIO10 Output Enable — When clear, bit[10] of the GPIO Output Data Register is enabled onto the GPIO[10] pin.
09	1 ₂	GPIO9 Output Enable — When clear, bit[9] of the GPIO Output Data Register is enabled onto the GPIO[9] pin.
08	1 ₂	GPIO8 Output Enable — When clear, bit[8] of the GPIO Output Data Register is enabled onto the GPIO[8] pin.
07	1 ₂	GPIO7 Output Enable — When clear, bit[7] of the GPIO Output Data Register is enabled onto the GPIO[7] pin.
06	1 ₂	GPIO6 Output Enable — When clear, bit[6] of the GPIO Output Data Register is enabled onto the GPIO[6] pin.
05	1 ₂	GPIO5 Output Enable — When clear, bit[5] of the GPIO Output Data Register is enabled onto the GPIO[5] pin.
04	1 ₂	GPIO4 Output Enable — When clear, bit[4] of the GPIO Output Data Register is enabled onto the GPIO[4] pin.
03	1 ₂	GPIO3 Output Enable — When clear, bit[3] of the GPIO Output Data Register is enabled onto the GPIO[3] pin.
02	1 ₂	GPIO2 Output Enable — When clear, bit[2] of the GPIO Output Data Register is enabled onto the GPIO[2] pin.
01	1 ₂	GPIO1 Output Enable — When clear, bit[1] of the GPIO Output Data Register is enabled onto the GPIO[1] pin.
00	1 ₂	GPIO0 Output Enable — When clear, bit[0] of the GPIO Output Data Register is enabled onto the GPIO[0] pin.



17.2.2 GPIO Input Data Register — GPID

The GPIO Input Data Register reflects the state of the appropriate **GPIO** bus pins following the deassertion of **P_RST#**.

Table 596. GPIO Input Data Register — GPID (Sheet 1 of 2)

Intel XScale® processor Local Bus Address offset +2484H		Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible
Bit	Default	Description
31:16	0000H	Reserved
15	GPIO[15] during P_RST# assertion	GPIO15 Input Data — This bit reflects the state of the GPIO[15] pin.
14	GPIO[14] during P_RST# assertion	GPIO14 Input Data — This bit reflects the state of the GPIO[14] pin.
13	GPIO[13] during P_RST# assertion	GPIO13 Input Data — This bit reflects the state of the GPIO[13] pin.
12	GPIO[12] during P_RST# assertion	GPIO12 Input Data — This bit reflects the state of the GPIO[12] pin.
11	GPIO[11] during P_RST# assertion	GPIO11 Input Data — This bit reflects the state of the GPIO[11] pin.
10	GPIO[10] during P_RST# assertion	GPIO10 Input Data — This bit reflects the state of the GPIO[10] pin.
09	GPIO[9] during P_RST# assertion	GPIO9 Input Data — This bit reflects the state of the GPIO[9] pin.
08	GPIO[8] during P_RST# assertion	GPIO8 Input Data — This bit reflects the state of the GPIO[8] pin.
07	GPIO[7] during P_RST# assertion	GPIO7 Input Data — This bit reflects the state of the GPIO[7] pin.
06	GPIO[6] during P_RST# assertion	GPIO6 Input Data — This bit reflects the state of the GPIO[6] pin.

**Table 596. GPIO Input Data Register — GPID (Sheet 2 of 2)**

Intel XScale® processor Local Bus Address offset +2484H Attribute Legend: RV = Reserved PR = Preserved RS = Read/Set RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible		
Bit	Default	Description
05	GPIO[5] during P_RST# assertion	GPIO5 Input Data — This bit reflects the state of the GPIO[5] pin.
04	GPIO[4] during P_RST# assertion	GPIO4 Input Data — This bit reflects the state of the GPIO[4] pin.
03	GPIO[3] during P_RST# assertion	GPIO3 Input Data — This bit reflects the state of the GPIO[3] pin.
02	GPIO[2] during P_RST# assertion	GPIO2 Input Data — This bit reflects the state of the GPIO[2] pin.
01	GPIO[1] during P_RST# assertion	GPIO1 Input Data — This bit reflects the state of the GPIO[1] pin.
00	GPIO[0] during P_RST# assertion	GPIO0 Input Data — This bit reflects the state of the GPIO[0] pin.



17.2.3 GPIO Output Data Register — GPOD

The GPIO Output Data Register is driven on a per bit basis on the appropriate **GPIO** bus pins following the deassertion of **P_RST#** when the corresponding bit in the GPOE register is cleared.

Table 597. GPIO Output Data Register — GPOD

Intel XScale® processor Local Bus Address offset +2488H Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible		
Bit	Default	Description
31:16	0000H	Reserved.
15	0 ₂	GPIO15 Output Data — This bit value is driven on the GPIO[15] pin when bit[15] of the GPOE register is cleared.
14	0 ₂	GPIO14 Output Data — This bit value is driven on the GPIO[14] pin when bit[14] of the GPOE register is cleared.
13	0 ₂	GPIO13 Output Data — This bit value is driven on the GPIO[13] pin when bit[13] of the GPOE register is cleared.
12	0 ₂	GPIO12 Output Data — This bit value is driven on the GPIO[12] pin when bit[12] of the GPOE register is cleared.
11	0 ₂	GPIO11 Output Data — This bit value is driven on the GPIO[11] pin when bit[11] of the GPOE register is cleared.
10	0 ₂	GPIO10 Output Data — This bit value is driven on the GPIO[10] pin when bit[10] of the GPOE register is cleared.
09	0 ₂	GPIO9 Output Data — This bit value is driven on the GPIO[9] pin when bit[9] of the GPOE register is cleared.
08	0 ₂	GPIO8 Output Data — This bit value is driven on the GPIO[8] pin when bit[8] of the GPOE register is cleared.
07	0 ₂	GPIO7 Output Data — This bit value is driven on the GPIO[7] pin when bit[7] of the GPOE register is cleared.
06	0 ₂	GPIO6 Output Data — This bit value is driven on the GPIO[6] pin when bit[6] of the GPOE register is cleared.
05	0 ₂	GPIO5 Output Data — This bit value is driven on the GPIO[5] pin when bit[5] of the GPOE register is cleared.
04	0 ₂	GPIO4 Output Data — This bit value is driven on the GPIO[4] pin when bit[4] of the GPOE register is cleared.
03	0 ₂	GPIO3 Output Data — This bit value is driven on the GPIO[3] pin when bit[3] of the GPOE register is cleared.
02	0 ₂	GPIO2 Output Data — This bit value is driven on the GPIO[2] pin when bit[2] of the GPOE register is cleared.
01	0 ₂	GPIO1 Output Data — This bit value is driven on the GPIO[1] pin when bit[1] of the GPOE register is cleared.
00	0 ₂	GPIO0 Output Data — This bit value is driven on the GPIO[0] pin when bit[0] of the GPOE register is cleared.

18.0 PMON Unit

18.1 PMON Counters

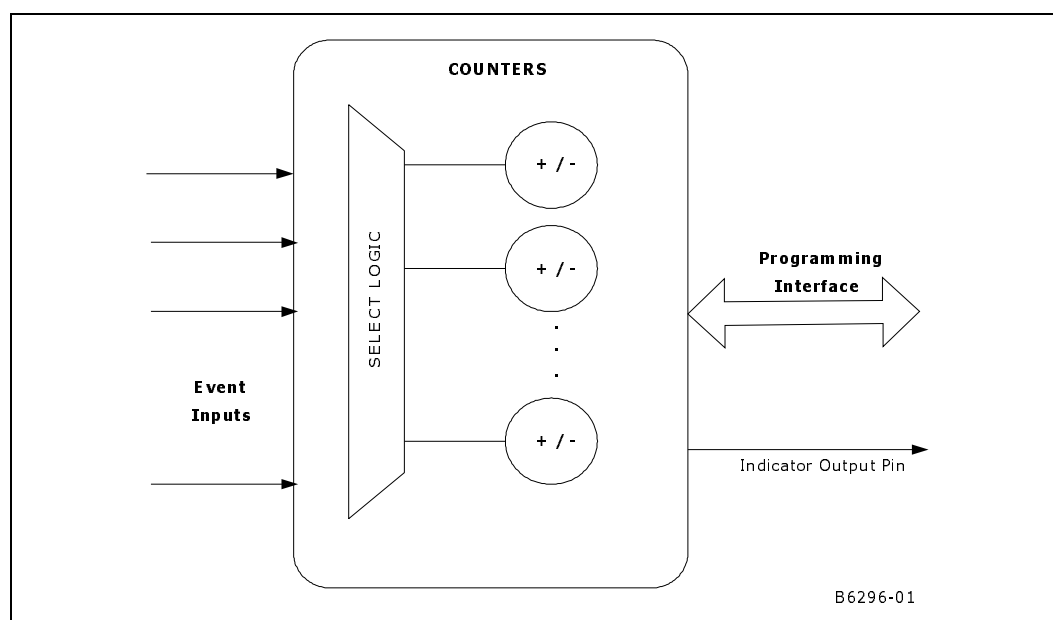
The Performance Monitoring (**PMON**) counters enable performance monitoring and gathering statistics of internal hardware events in real-time. This implementation provides users with direct event counting and timing for performance monitoring and system debugging purposes. It provides enough visibility into the internal architecture to perform utilization studies, workload characterization, and application tuning.

18.2 Overview

At the heart of the **PMON** functionality are counters with associated registers. Each counter has a corresponding command, event, status, and data register. The **PMON** unit implements eight counters.

Signals representing events from throughout the chip are routed to the **PMON** unit. Software can select which events are recorded during a measurement session. The starting, stopping, and sampling of the counters can be controlled by either software or hardware. This can be done in a time-based (polling) or event-based fashion. Each counter can be incremented or decremented by different events. In addition to simple counting of events the unit can provide data for histograms, queue analysis, and conditional event counting (example: How many times did event A happen before the first event B took place).

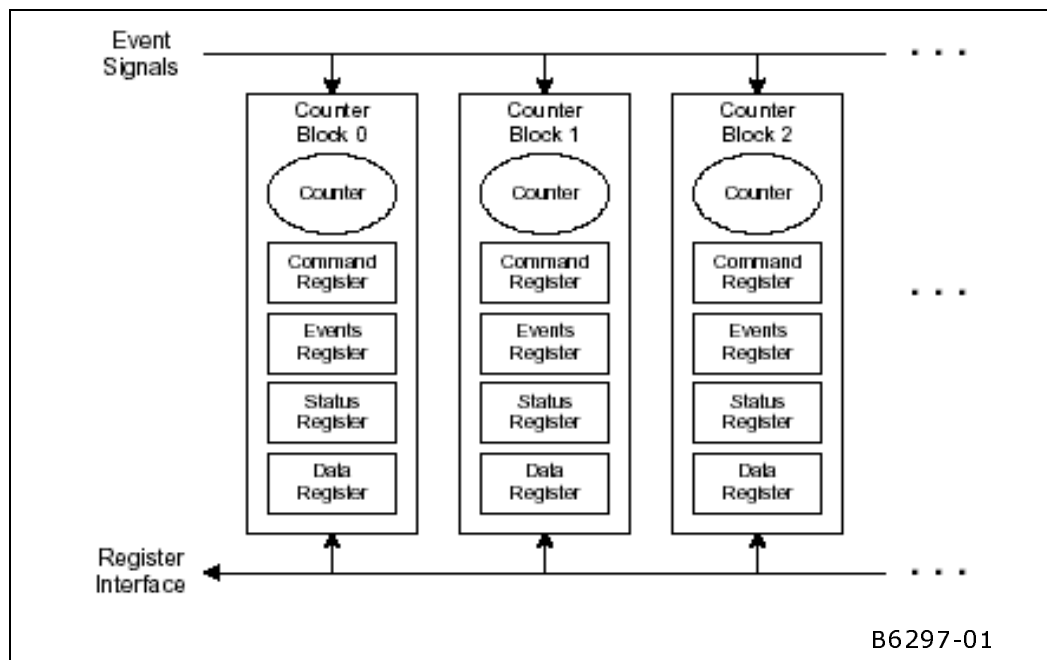
Figure 155. Example Block Diagram of Component with Counter



18.2.1 Clock Counter Control

When a counter is sampled, the current value of the counter is latched into the corresponding data register. The command, event, status, and data registers are accessible via memory mapped registers in order to facilitate high-speed sampling.

Figure 156. Conceptual Diagram of Counter Array



18.3 Definitions

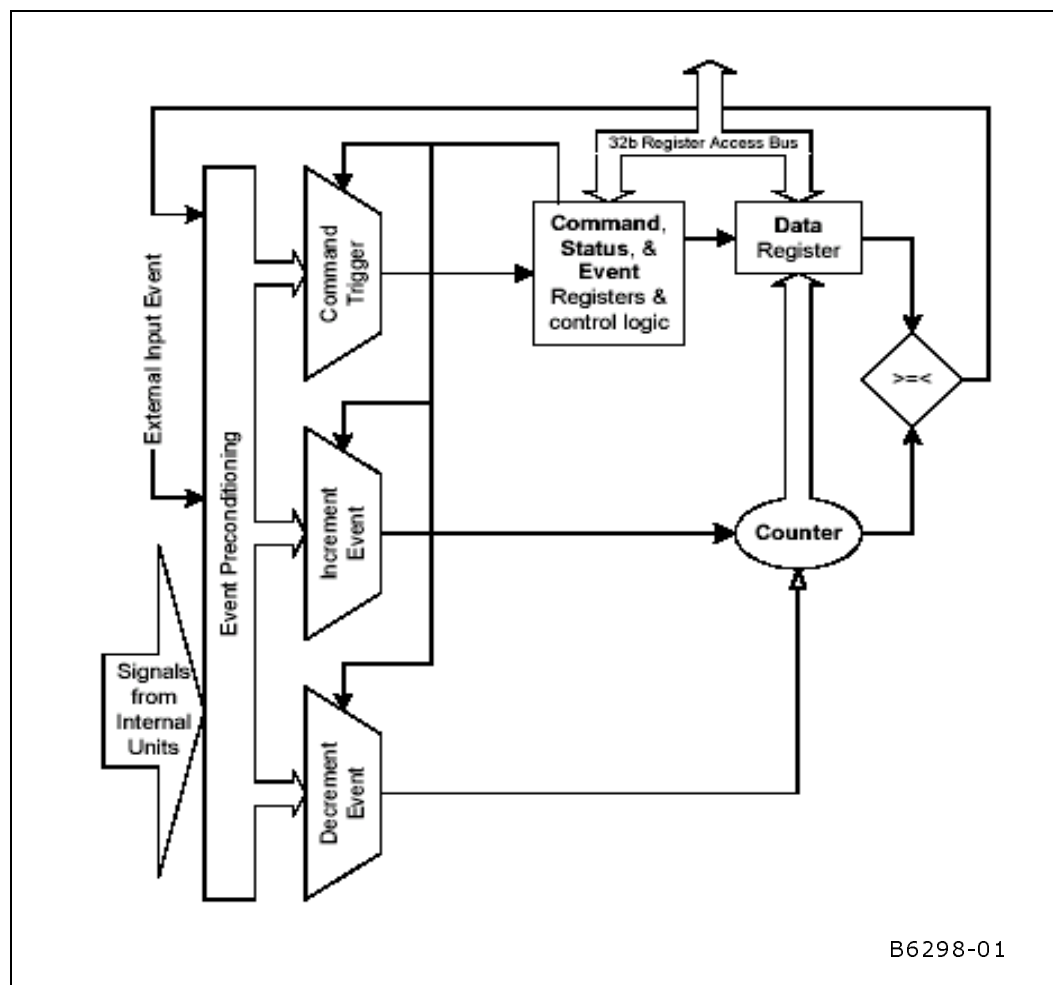
- Duration Count - The counter is incremented for each clock for which the event signal is asserted logic high.
- Occurrence Count - The counter is incremented each time a rising edge of the event signal is detected.
- Preconditioning - Altering a signal that represents an event before it is presented to be counted by the **PMON** unit. This includes clock crossing logic.

Two optional external pins allow for external visibility and control of the counters. The output pin signals that one of the following conditions generated an interrupt from any one of the counters:

- a programmable threshold condition was true,
- a command was triggered to begin
- a counter overflow or underflow occurred.

The figure below represents a single counter block. The muxes, registers, and all other logic is repeated for each counter that is present. There is a threshold event from each counter block that feeds into each mux.

Figure 157. Example Block Diagram of Single PMON Counter



18.4 Data Collection

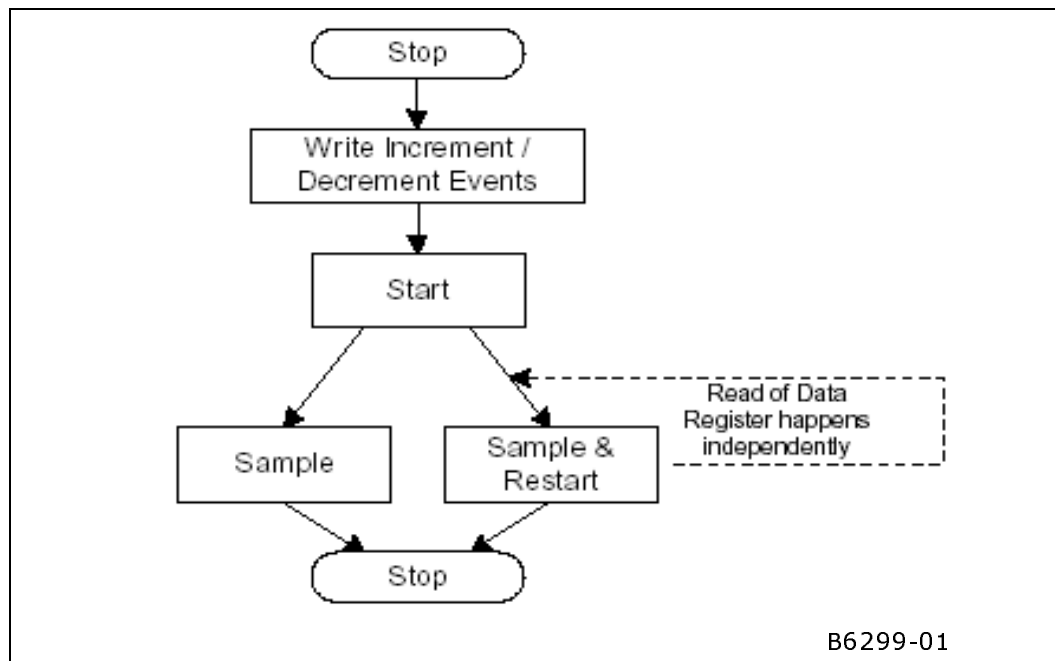
The following sections provide some insight into the intended use of the **PMON** counters at a very low level. The examples and accompanying explanation should prove especially valuable in the creation of test cases for both hardware and software.

The hardware face to the **PMON** counters is not intended to be any wider than 32 bits. This means that all registers are accessed one at a time.

All of the following examples assume starting with an idle system (all counters stopped and all registers set to default values). The waveforms in the examples do not explicitly always show when each **PMON** command was executed, but the reader should be able to deduce when each command would have or could have executed. For the purposes of these examples, the waveforms treat the counters as being level sensitive to the appropriate events.

18.4.1 Time Based Sampling

Figure 158. Flowchart of Example Commands Sequence





Example 17 has been simplified by using 12 clocks as the sampling period. In a real system the sampling would more likely be something like 1 ms. There is a certain amount of overhead associated with writing and reading to any **PMON** registers. The more frequent the interaction between the **PMON** counters and any software, the larger the margin of error that is injected into the final results.

The sampling period can be controlled by a CPU counter or another **PMON** counter periodically interrupting the system and allowing the software to read data registers or do whatever else may be desired.

Example 17. Simple Counting

This example demonstrates how to measure the number of times event A occurs during 12 clocks.

Table 598. Simple Time Based Counting of Events Example

Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Event Register	0	Event A	None (000h)	
Start	0			Immed (000h)
* Sample & Restart	0			Immed (000h)
Read Data Register	0			
-- Wait 12 clocks before returning to * --				

An alternative way to represent the data in the preceding table is as follow:

Write Event Register 0 (Increment = Event A)

Start Counter 0 immediately

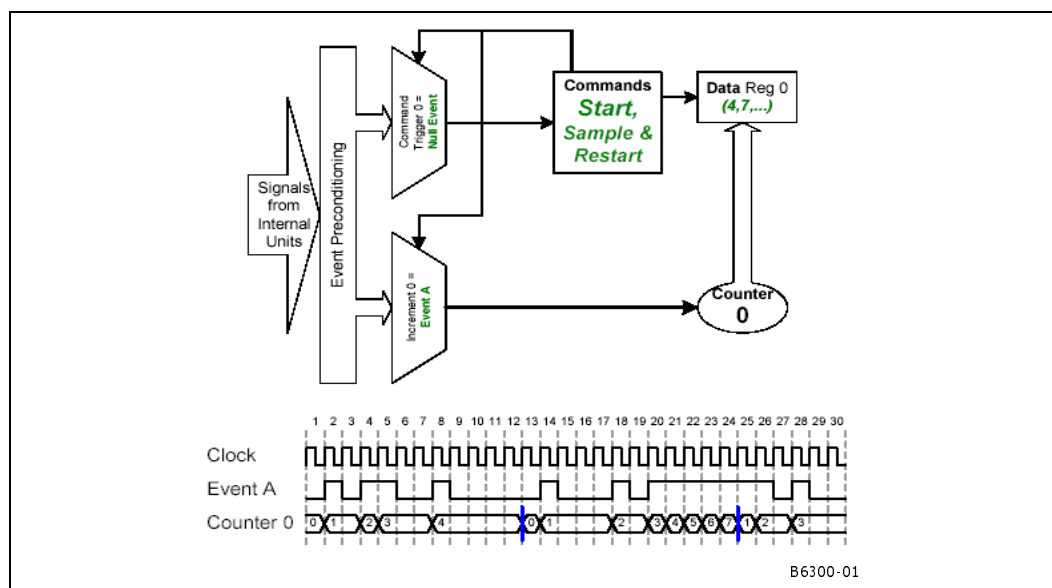
Repeat every 12 clocks

Sample & Restart Counter 0 (Threshold Condition Code is NA)

Read Data Register 0

End Repeat

Figure 159. Block Diagram and Waveforms of Time Based Sampling Example





18.4.2 Hardware Event Based Control

Hardware event based control allows a hardware event to control when another command is executed. An example of this is controlling when a sample (snapshot) is taken of the active counter(s). This is required to facilitate, among other things, hardware **data queue analysis**. No command is executed until the *command trigger mux* detects the event in the command trigger field of the command register. This allows start, stop, sample, and other commands to be executed as a result of other events happening.

Command Triggers refers to the ability of a command to be issued to the **PMON** unit, and have it not be executed until the desired event, as programmed when the pending command was issued, is detected.

The **PMON** unit has no ability to queue commands so programming must be written so as not to overwrite pending commands unless that is the desired effect.

For example, one could count how many memory reads happen before the first cache hit occurs. This would be accomplished by programming the desired **PMON** counters to start counting memory read events (Event A in [Example 18 on page 953](#)). Immediately following that command the **PMON** controller is programmed to stop counting the memory read events.



However, execution of this command is conditional upon a cache hit occurring (Event B in [Example 18](#)). Thus, the cache hit command becomes the “Command Trigger” that must occur before the memory read events would no longer be counted.

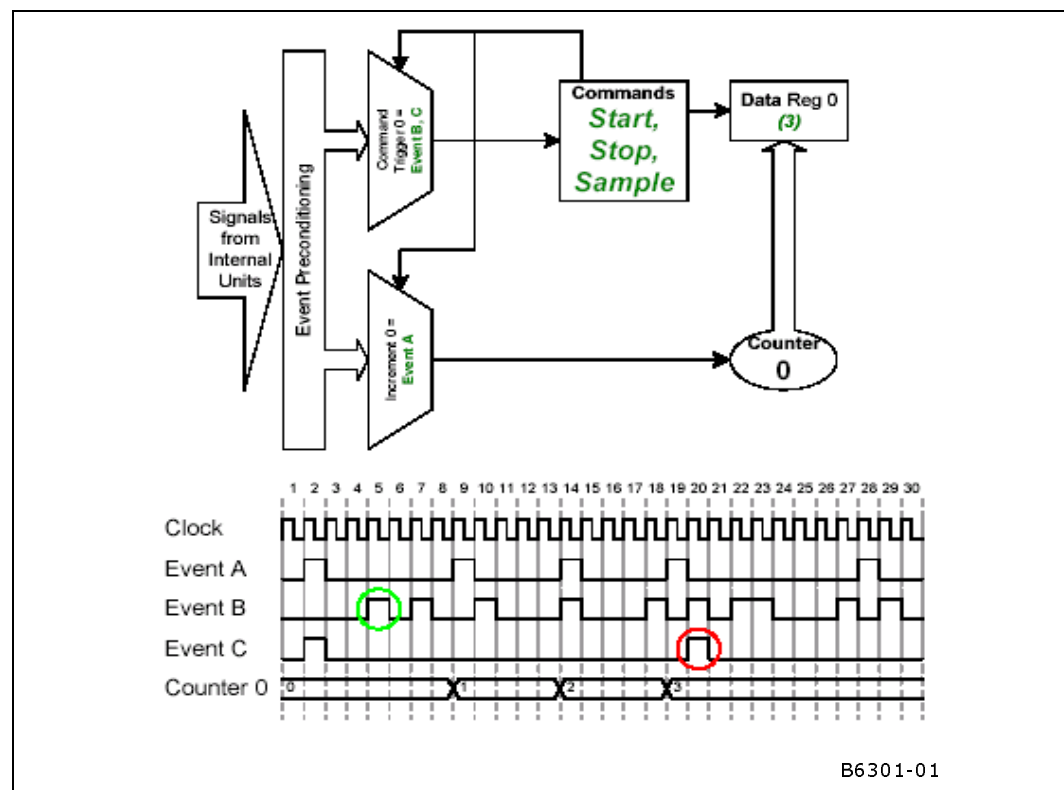
Example 18. How many Event A’s happen before the first Event B is detected

This example demonstrates how to measure the number of times event A occurs before the first occurrence of event B.

Table 599. Hardware Event Based Event Counting Example

Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Event Register	0	Event A	None (000h)	
Start	0			Immed (000h)
Stop	0			Event B
Sample	0			Immed (000h)
Read Data Register	0			

Figure 160. Block Diagram and Waveforms of Time Based Sampling Example

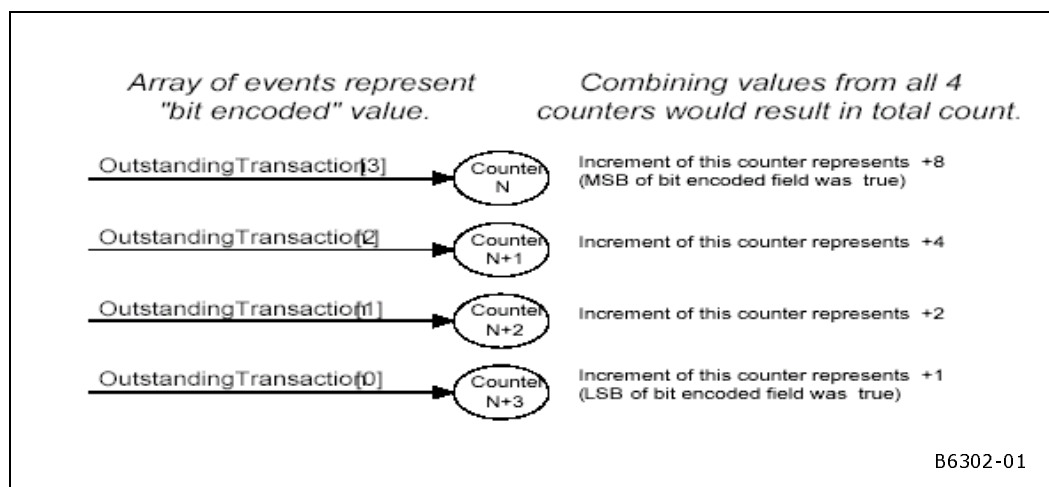


18.4.3 Incrementing By More Than 1

For scenarios where it is desired to increment a count by more than one on a single clock tick, the **PMON** unit can be sent a "bit encoded" value. This allows the counters to track an increment/decrement of up to 2^{N-1} each clock tick, where N is the number of counters available. For example, with 8 counters we could track an increment/decrement of up to 128 (2^7) each clock.

This would be done by assigning an array of events to represent a particular value to count. The counters would be coordinated to mimic a single counter which by asserting the appropriate event signals cause the set of counters to be incremented and/or decremented by the in such a way that post processing of the counter values.

Figure 161. Block Diagram & Waveforms of Time Based Sampling Example





18.4.4 Queue Analysis

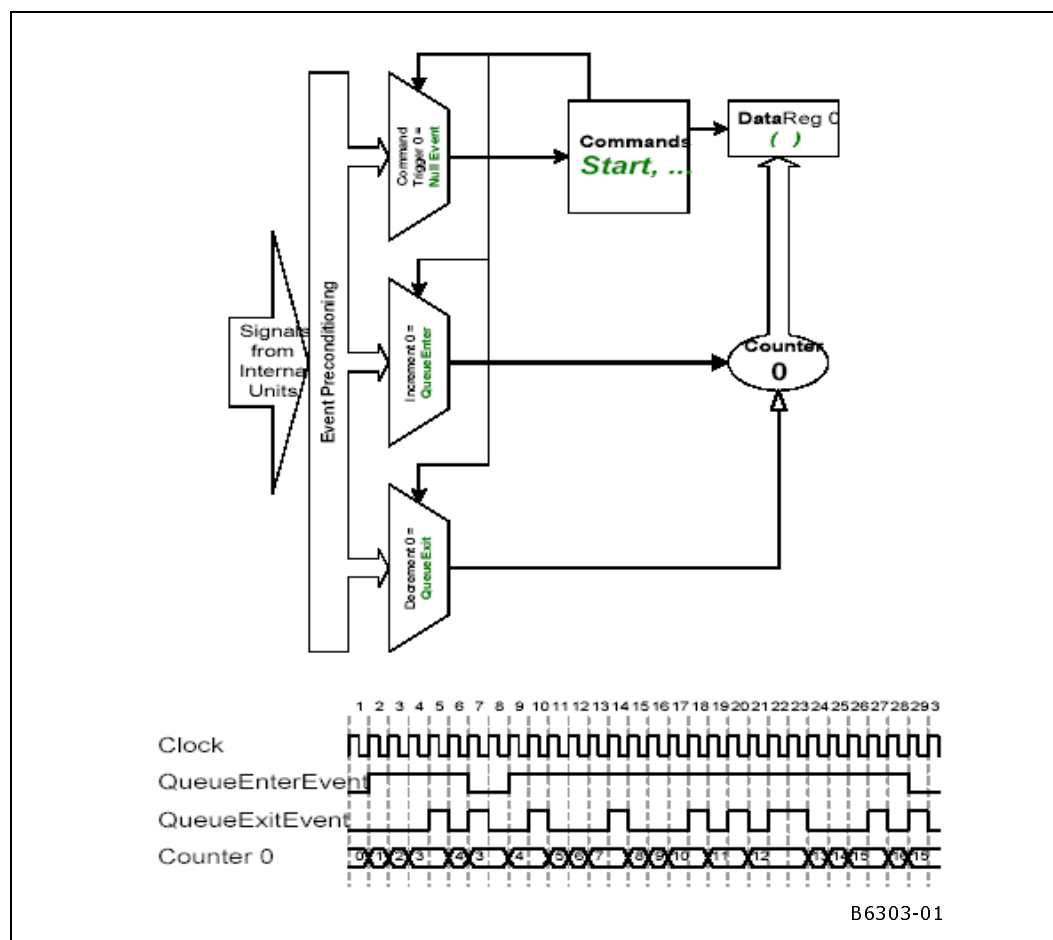
Example 19. Queue Depth Example

This example demonstrates how to measure the current fill level (depth) of a particular queue.

Table 600. Hardware Event Based Event Counting Example

Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Event Register	0	Queue enter	Queue Exit	
Start (CC is =)	0			Immed (000h)
Sample	0			Immed (000h)
Read Data Register	0			

Figure 162. Block Diagram & Waveforms of Time Based Sampling Example



Gathering queue depth information, as in [Example 19](#), is probably more useful when it is presented in a histogram. [Example 20 on page 956](#) outlines the command sequences required to generate such a histogram. See the Head of Queue Histogram example in another section for more details on how to create histograms.

**Example 20. Queue Depth Histogram Example**

This example demonstrates how to measure the current fill level (depth) of a particular queue.

Table 601. Queue Depth Histogram Example

Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Threshold (bucket size) in Data Register	0			
Write Event Register	0	Queue Enter	Queue Exit	
Write Event Register	1	Threshold Event 0		
Start (CC is =)	0			Queue Empty
Start	1			Immed (000h)
-- Run Workload --				
Sample	1			Immed (000h)
Read Data Register	0			
-- Repeat entire sequence with new bucket size --				

No block diagram for this example.



Example 21. Head of Queue Histogram Example

A histogram can be created using two counters working together. The following applies to all of the histogram examples:

- Start with idle system (reset counters).
- Only one slice of a histogram is measured per experiment, therefore histograms can only be generated for repeatable workloads.
- Repeat these steps, changing threshold value and using an unchanging repeatable workload to generate data for histogram.

This example demonstrates how often has a pre-amount of time elapsed when a certain event takes place.

Table 602. Head of Queue Histogram Example

Opcode	Target Counter	Increment Event	Decrement Event	Trigger Event
Write Threshold (bucket size) in Data Register	0			
Write Event Register	0	Queue Not Empty		
Write Event Register	1	Threshold Event 0		
Start	0			Queue Empty
Restart (CC is =)	0			Queue Exit
Start	1			Immed (000h)
-- Run Workload --				
Sample	1			Immed (000h)
Read Data Register	0			
-- Repeat entire sequence with new bucket size --				

An alternative way to represent the data in the preceding table is as follow:

```

For X=1 to 3
  Write Threshold (bucket size = X) into Data Register 0
  Write Event Register 0 (Increment = QueueNotEmpty)
  Write Event Register 1 (Increment = Threshold Event 0)
  Start Counter 0 immediately
  Restart Counter 0 (with Threshold Condition Code set to =)
    whenever QueueExitEvent triggers it
  Start Counter 1 immediately
  Run workload
  Sample Counter 1
  Read Data Register 1
End For

```

Figure 163. Block Diagram of HOQ Histogram Example

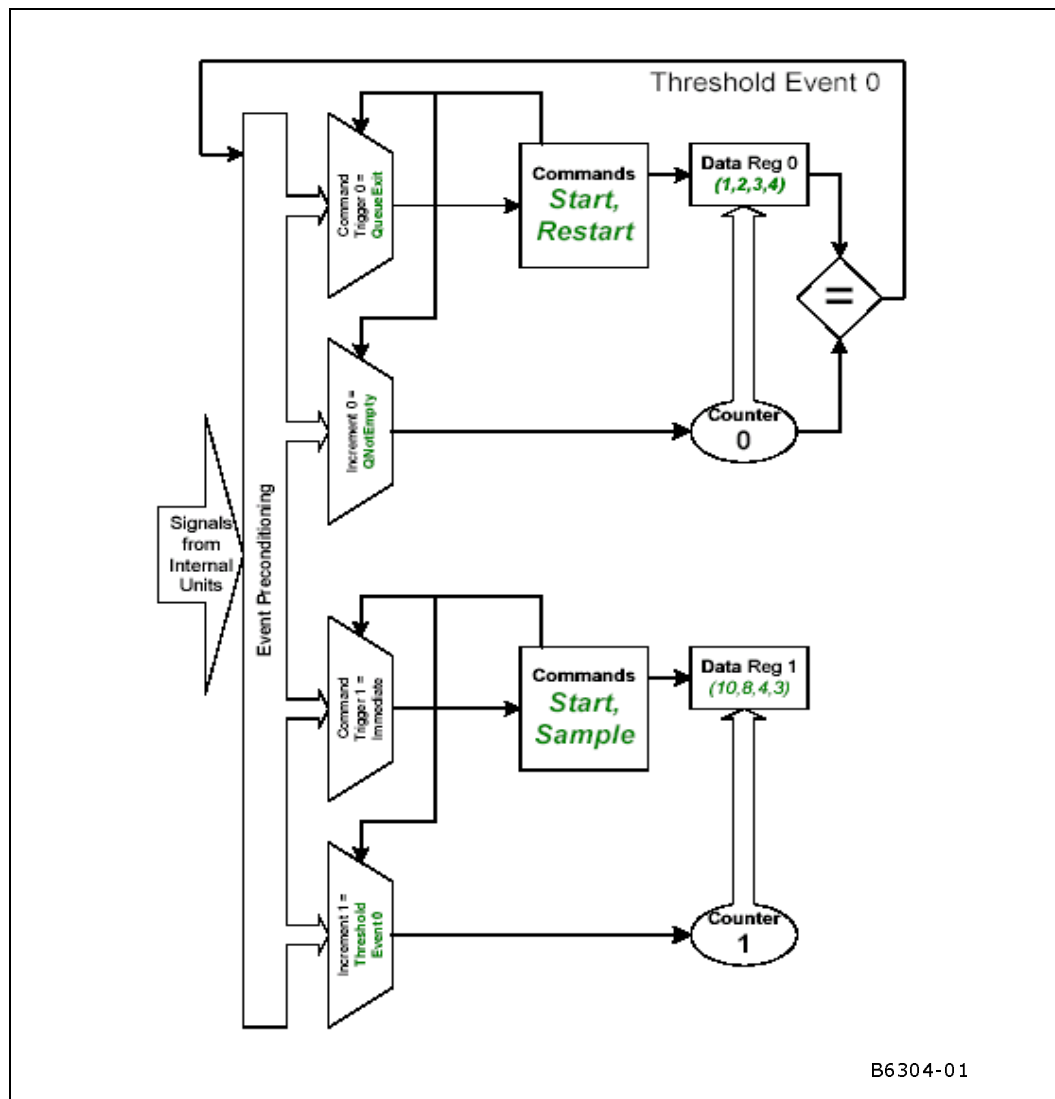
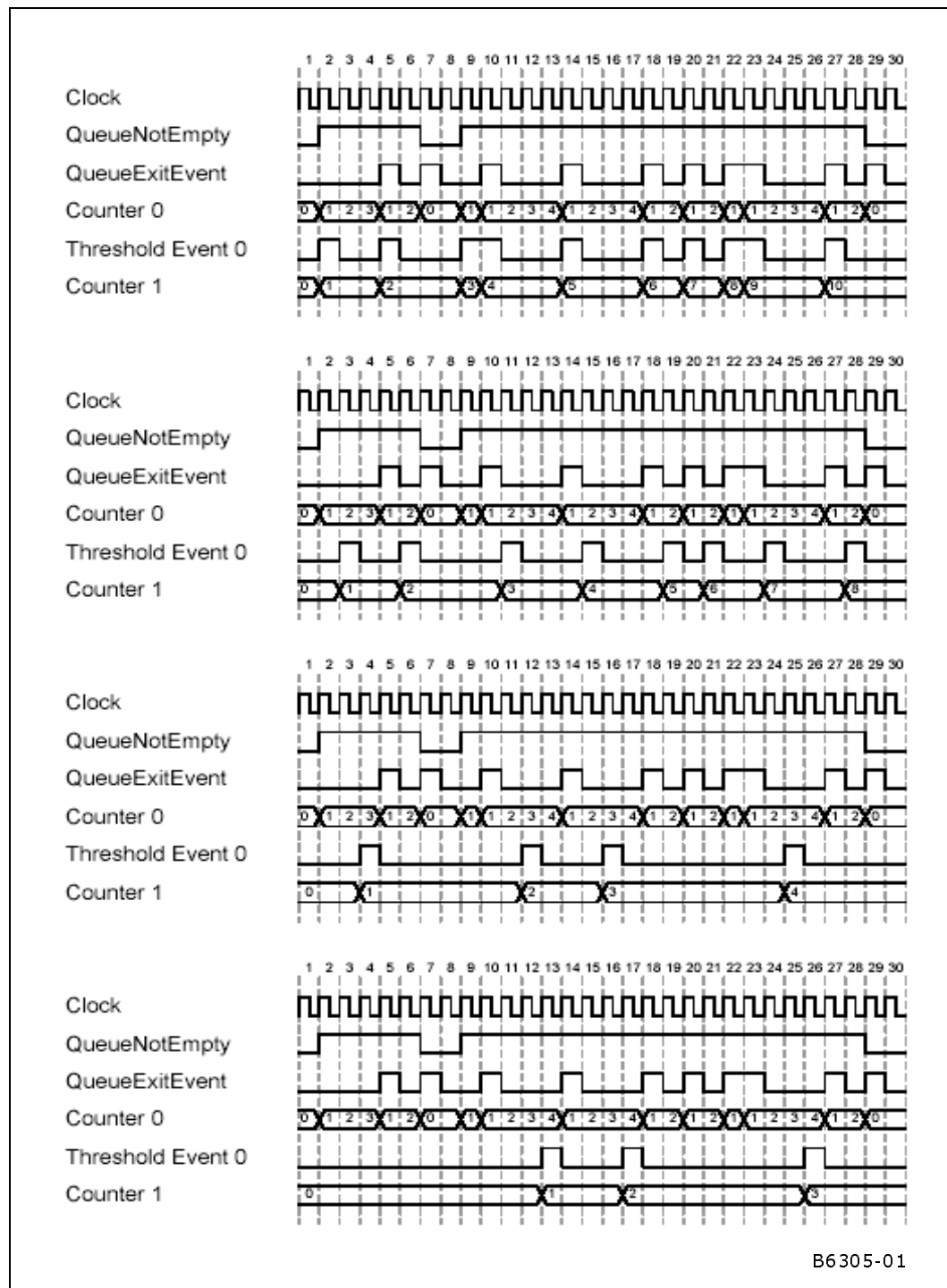




Figure 164. Waveforms of HOQ Histogram Example



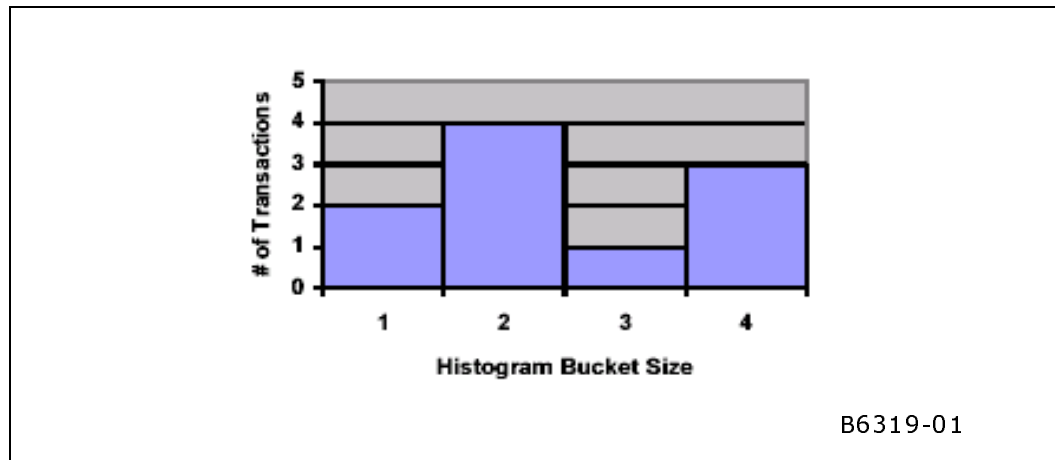
In each of the 4 iterations above, the exact same repeatable workload produces identical queue behavior, but the Counter 1 value (called Condition Code Matches below) differs due to having a different threshold value each time. The condition code match values (10, 8, 4, 3) from the 4 iterations can be used to compute the values of a histogram as shown in the following two figures.

Figure 165. Processing of HOQ Histogram Example

Threshold Value (Bucket Size)	Condition Code Matches	Cumulative Transactions	# of Transactions
4:	3	0	3
3:	4	3	1
2:	8	4	4
1:	10	8	2

B6318-01

Figure 166. Output from HOQ Histogram Example





18.5 Non-Register-Based Interfaces

This section describes the interfaces to the **PMON** unit that are not part of the register scheme already documented.

18.5.1 Events Input Port

Signals representing internal events are sent to the event preconditioning block where they are conditioned when required. The most common preconditioning is likely to be clock synchronization. No programmable Boolean operations can be performed on these events, but appropriate Boolean operations can be performed by the hardware.

Duration type events continually assert their signal high ('1'). The event preconditioning block 'ANDs' the duration type signal with a clock to produce the correct count.

Any events from different frequency domains must be preconditioned to assure count accuracy measured. For these clock domain crossing signals, 95% accuracy is sufficient over a 1 us or larger sampling window. It would be very difficult to assure accuracy of very small windows (< 1 us) without requiring a great amount of hardware to verify.

18.5.2 Output Signals

There are three potential sources of internal indicators from each counter. Each source can independently generate an indication. These are:

- a programmable threshold condition was true,
- a command was triggered to begin
- a counter overflow or underflow occurred.

Each of these conditions always sets a corresponding status bit. Indicator enable bits control which of these indicator sources drive the top level indicator. The indicators are OR'd together as shown in [Figure 167, "Indicator Tree" on page 962](#). These internal signals are further controlled to generate either an Interrupt or an Indicator Output.

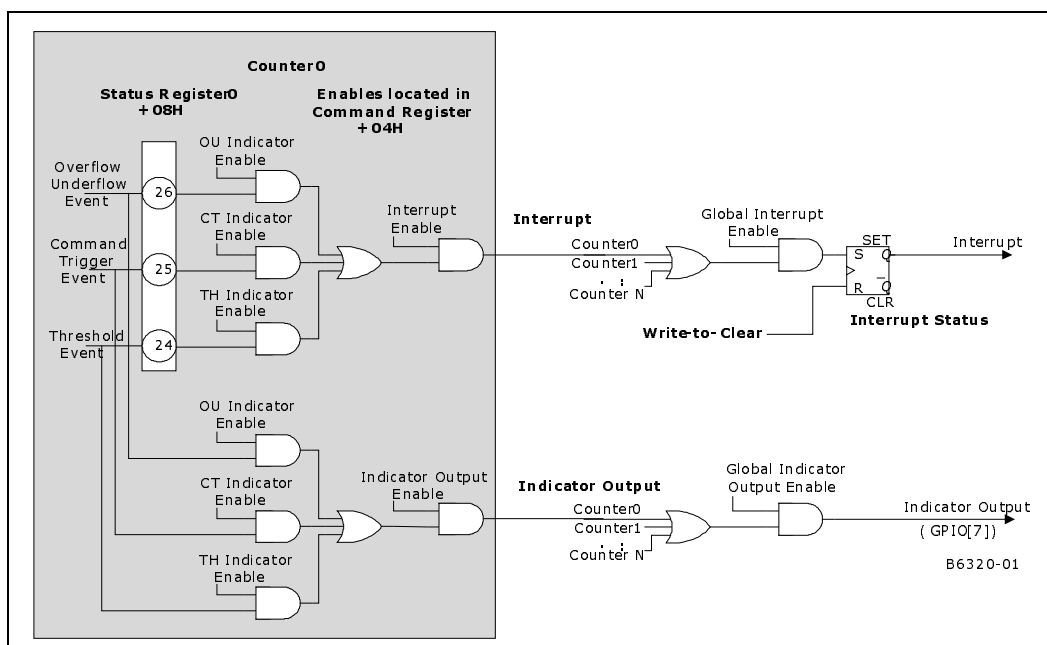
18.5.2.1 Indicator Output

The **PMONOUT** pin is a shared indicator output pin. The resulting signal allows external elements (OS, drivers, logic analyzer, etc.) to be aware of indicators without having to rely on interrupts.

For 81341 and 81342, the **PMONOUT** function is multiplexed onto the GPIO7 pin. When the **PMON** Indicator Output Enable bit is set in the “**PMON Feature Enable Register - PMONEN**”, the **PMON** output function overrides the GPIO7 setting in the Section 17.2.1, “**GPIO Output Enable Register — GPOE**” on page 943 and Section 17.2.3, “**GPIO Output Data Register — GPOD**” on page 946.

Warning: Since GPIO7 could be used for other purposes, care must be taken when enabling the **PMON** Indicator Output.

Figure 167. Indicator Tree



Note: Indicator and Interrupt Enables are located in the “**PMON Command Register 0-7 - PMON_CMD[0:7]**” on page 967
Global Indicator and Interrupt Enables are located in the “**PMON Feature Enable Register - PMONEN**” on page 964

18.5.2.2 Interrupt Output

An internal interrupt is delivered to the Interrupt Control Unit when:

- Interrupts are enabled in the “**PMON Feature Enable Register - PMONEN**” (PMONEN[0]=1).
- One or more of the interrupt sources are enabled within the individual counter via the indicator enable bits.
- The interrupt enable bit is set within the individual counter
- One or more of the enabled indicators is true

This sets the internal interrupt generated bit in the **PMONSTAT** register and asserts the **PMON** interrupt input to the Interrupt control unit.



18.5.3 Internal Bus Addresses

The Internal Bus Address Offset to PMMRBAR of any **PMON** Register can be derived by adding the 4 KB address aligned Internal Bus Memory Mapped Register Range Offset (Table 603, “PMON Internal Bus Memory Mapped Register Range Offsets” on page 963) to the Register Offset (Table 604, “PMON Register Summaries” on page 963)

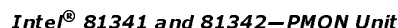
For example the offset to PMMRBAR of the “PMON Status Register - PMONSTAT” would be (4 E000H+044H) or 4 E044H.

Table 603. PMON Internal Bus Memory Mapped Register Range Offsets

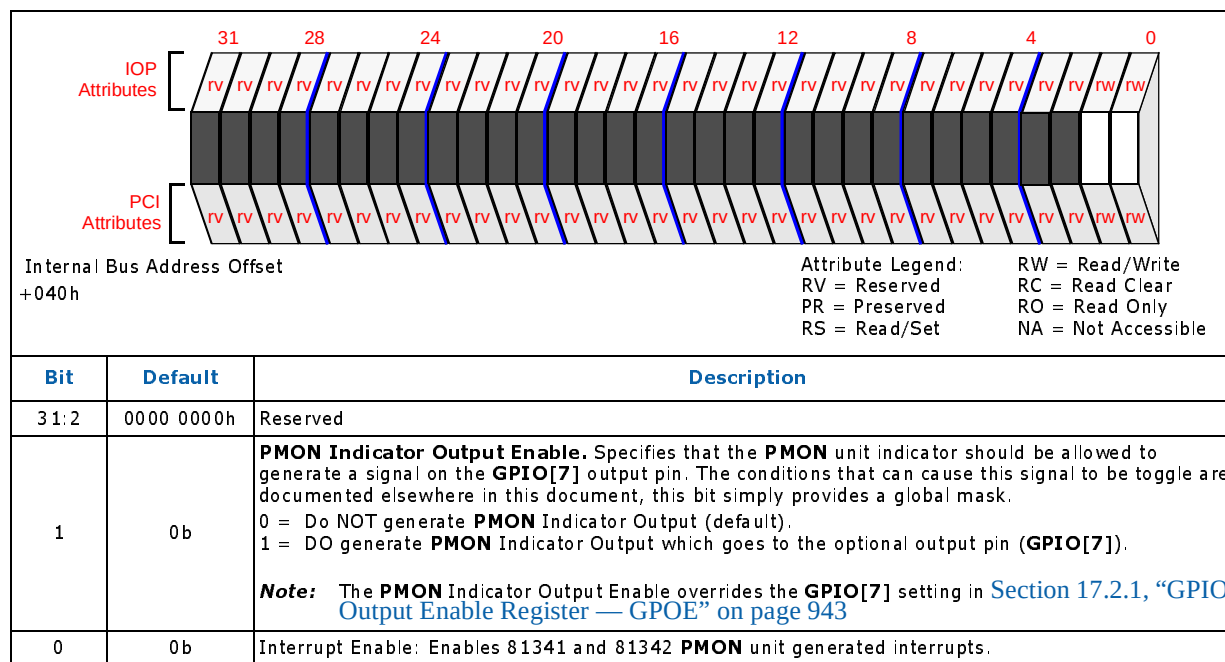
Internal Bus MMR Address Range Offset (Relative to PMMRBAR)
+4 E000H

Table 604. PMON Register Summaries

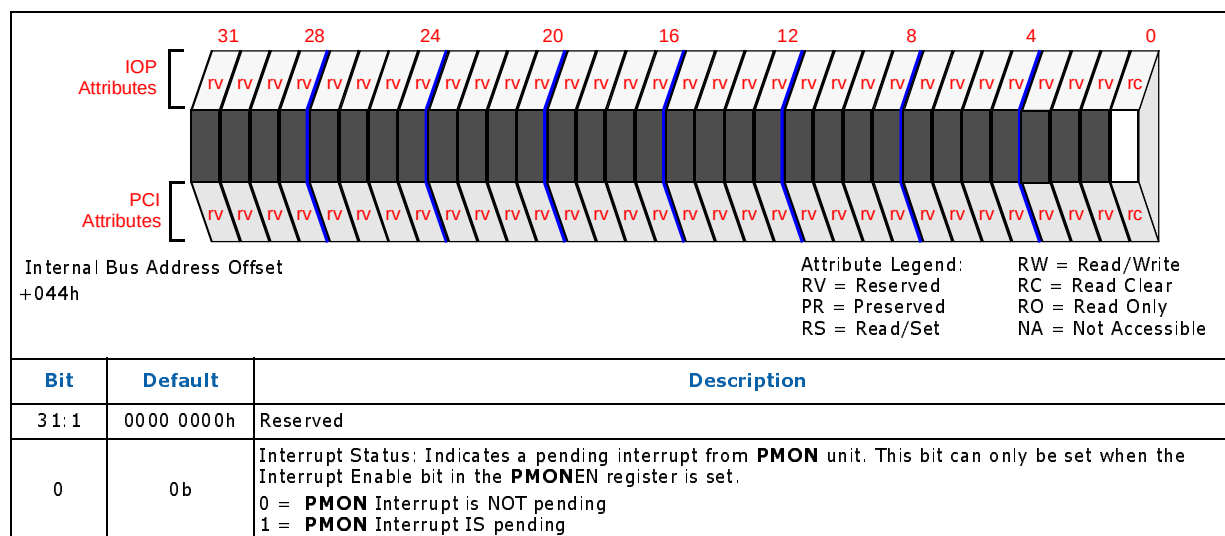
Register Offset	Register Name
+040h	PMON Feature Enable Register - PMONEN
+044h	PMON Status Register - PMONSTAT



Contains control bits for **PMON** unit.



Contains status bits for **PMON** unit.





18.5.6 PMON Memory Mapped Registers

The memory mapped registers of **PMON** unit are accessible by the Intel XScale® core.

The first set of registers provide the control of the **PMON** unit for selecting events to monitor and for data sampling. Each counter has one command, one events, one status, and one data register associated with it. These registers are numbered 0 through 7.

The location of these registers are specified as a relative offset to a 512KB aligned global PMMR offset. The default for the 512KB aligned offset is 0 FFD8 0000H defined by the PMMRBAR register. See also [Chapter 21.0, "Peripheral Registers"](#).

The Internal Bus Address Offset to PMMRBAR of any **PMON** Register can be derived by adding the 8 KB address aligned Memory Mapped Register Range Offset ([Table 607, "PMON Internal Bus Memory Mapped Register Range Offsets" on page 965](#)) to the Register Offset ([Table 608, "PMON Register Summaries" on page 966](#))

For example the offset to PMMRBAR of the "**PMON** Command Register 0" would be (1 A000H+004H) or 1 A004H.

Table 607. PMON Internal Bus Memory Mapped Register Range Offsets

PMON Memory Mapped Address Range Offset (Relative to PMMRBAR)
+1 A000H



Table 608. PMON Register Summaries

Register Offset	Register Name
+000h	PMON Command Register 0 - PMON_CMD0
+004h	PMON Event Register 0 - PMON_EVR0
+008h	PMON Status Register 0 - PMON_STS0
+00Ch	PMON Data Register 0 - PMON_DATA0
+010h	PMON Command Register 1- PMON_CMD1
+014h	PMON Event Register 1- PMON_EVR1
+018h	PMON Status Register 1- PMON_STS1
+01Ch	PMON Data Register 1 - PMON_DATA1
+020h	PMON Command Register 2- PMON_CMD2
+024h	PMON Event Register 2- PMON_EVR2
+028h	PMON Status Register 2- PMON_STS2
+02Ch	PMON Data Register 2 - PMON_DATA2
+030h	PMON Command Register 3 - PMON_CMD3
+034h	PMON Event Register 3- PMON_EVR3
+038h	PMON Status Register 3- PMON_STS3
+03Ch	PMON Data Register 3 - PMON_DATA3
+040h	PMON Command Register 4 - PMON_CMD4
+044h	PMON Event Register 4 - PMON_EVR4
+048h	PMON Status Register 4 - PMON_STS4
+04Ch	PMON Data Register 4 - PMON_DATA4
+050h	PMON Command Register 5 - PMON_CMD5
+054h	PMON Event Register 5 - PMON_EVR5
+058h	PMON Status Register 5 - PMON_STS5
+05Ch	PMON Data Register 5 - PMON_DATA5
+060h	PMON Command Register 6 - PMON_CMD6
+064h	PMON Event Register 6 - PMON_EVR6
+068h	PMON Status Register 6- PMON_STS6
+06Ch	PMON Data Register 6 - PMON_DATA6
+070h	PMON Command Register 7 - PMON_CMD7
+074h	PMON Event Register 7- PMON_EVR7
+078h	PMON Status Register 7- PMON_STS7
+07Ch	PMON Data Register 7- PMON_DATA7
+080h through +FFFh	Reserved



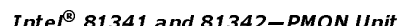
18.5.6.1 PMON Command Register 0-7 - PMON_CMD[0:7]

This 32-bit register allows control of the **PMON** counter. When this register is written, the previous register contents are overwritten. All 32 bits must be programmed each time the register is written. When the register contained a command that was still waiting to be triggered, it would be flushed without ever being executed. The currently executing command continues to be executed only until the newly programmed command is triggered to execute.

Table 609. PMON Command Register 0-7 - PMON_CMD[0:7] (Sheet 1 of 4)

Register Offset		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
PMON_CMD0		+000h																											
PMON_CMD1		+010h																											
PMON_CMD2		+020h																											
PMON_CMD3		+030h																											
PMON_CMD4		+040h																											
PMON_CMD5		+050h																											
PMON_CMD6		+060h																											
PMON_CMD7		+070h																											

Bit	Default	Description
31:29	00b	Reserved
28	0b	Interrupt Enable This bit causes the counter to set the global interrupt generated bit in the PMONSTAT register. 0 = An interrupt is not generated from this counter 1 = The global interrupt generated bit is set when one or more of the Overflow/Underflow, Command Trigger, and Threshold status bits are asserted and the individual indicator enable(s) are set. An interrupt is delivered when there is not an existing interrupt being processed from PMON . Upon servicing the Interrupt, the status registers should be cleared by software so that another Interrupt is not sent.
27	0b	Indicator Output Enable: 0 = PMONOUT is not asserted when the indicators are valid. 1 = The PMONOUT signal is asserted when the conditions enabled via the indicator enable bits are true.
26	0b	Overflow/Underflow Indicator Enable (OUIE) 0 = No indication provided when a counter overflow or underflow occurs except for setting the Overflow/Underflow Indicator (OUI) status bit. 1 = When the overflow/underflow condition is true, the counter sets the status bit and possibly assert the PMONOUT pin or enable the generation of an interrupt. This is controlled by the PMONOUT Enable and Interrupt Enable respectively. Interrupts are driven directly off of the status bits, whereas the PMONOUT pin is driven directly from the condition.
25	0b	Command Trigger Indicator Enable (CTIE) 0 = No indication provided when a command is triggered except for setting the Command Trigger Indicator (CTI) status bit. 1 = When the command trigger condition is true, the counter sets the status bit and possibly assert the PMONOUT pin or enable the generation of an interrupt. This is controlled by the PMONOUT Enable and Interrupt Enable respectively. Interrupts are driven directly off of the status bits, whereas the PMONOUT pin is driven directly from the condition. Note: The null event does not set the Command Trigger Indicator in the PMON_STS register, nor does it cause a Command Trigger that is reflected by the PMONOUT signal.



Bit	Default	Description
24	0b	Threshold Indicator Enable (THIE) 0 = No indication provided when threshold condition is true except for setting the Threshold Indicator (TI) status bit. 1 = When the threshold condition is true, the counter sets the status bit and possibly assert the PMONOUT pin or enable the generation of an interrupt. This is controlled by the PMONOUT Enable and Interrupt Enable respectively. Interrupts are driven directly off of the status bits, whereas the PMONOUT pin is driven directly from the condition.
23:21	000b	Condition Code (CC) This field contains the code that indicates what type of threshold compare is done between the counter and the data register. For all non-0 values of this field, the counter's data register contains the threshold value. The outcome of this compare generates a threshold event and potentially an interrupt when that capability is enabled. - Bit 23 is for less than (<) - Bit 22 is for equal (=) - Bit 21 is for greater than (>) Select the proper bit mask for desired threshold condition: 000 = False (no threshold compare) 001 = Greater Than 010 = Equal 011 = Greater Than or Equal 100 = Less Than 101 = Not Equal 110 = Less Than or Equal 111 = True (always generate threshold event)
20	0b	Select ALL Counters (SAC). This bit controls how the opcode in bits 19:16 is applied to all counters. The rest of the PMON_CMD register is not affected by the setting of this bit. 0 = The opcode is applied only to the counter associated with this command register. 1 = The opcode is applied to ALL counters. This means that every command register is written to with the same value that was written to this particular command register. This is particularly useful for resetting all counters with a single command or starting or stopping all counters simultaneously. Note: This bit is only valid in Command Register 0 and has no effect in non-0 command registers. "Globally" executed commands (by setting this bit in Command Register 0) always override "locally" executed commands.



Table 609. PMON Command Register 0-7 - PMON_CMD[0:7] (Sheet 3 of 4)

Register Offset																Attribute Legend:													
PMON_CMD0	+000h															RW = Read/Write													
PMON_CMD1	+010h															RV = Reserved													
PMON_CMD2	+020h															PR = Preserved													
PMON_CMD3	+030h															RS = Read/Set													
PMON_CMD4	+040h															RC = Read Clear													
PMON_CMD5	+050h															RO = Read Only													
PMON_CMD6	+060h															NA = Not Accessible													
PMON_CMD7	+070h																												

Bit	Default	Description
19:16	0h	Opcode 0000 = Stop. The corresponding counter does not count. 0001 = Start. The corresponding counter begins counting. Each counter increments by one when the corresponding increment event occurs or decrements by one when the corresponding decrement event occurs. All duration type events toggle every PMON unit clock tick that the event is true. The desired increment and decrement events must be selected before this command executes. 0010 = Sample. The corresponding counter value is latched into the corresponding data register, which can then be read by reading the appropriate data register. The counter continues to count without being reset. When the Condition Code is NOT False then the Data Register is not written when a sample takes place. In other words, the Data Register is only ever updated with counter value when the Condition Code is False. 0100 = Reset. The corresponding counter is reset to 0000 0000h and stops counting. The 32 bit wide counter allows 4 billion clock ticks or occurrences to be counted between sample commands. When the counter rolls over, the overflow status bit is set in the corresponding status register. 0101 = Restart. The corresponding counter resets, then starts counting again. This is essentially a Reset & Start command. This functionality facilitates generating histograms by allowing an event to trigger to clear the counter and resume counting with no further intervention. 0110 = Sample & Restart. The Sample command happens and is followed immediately by the Restart command. 1111 = Preload. The corresponding counter is set to the value that is located in the associated data register. This facilitates rollover and overflow validation. The counter remains in the same state when preloaded. When the counter was counting before the preload was executed it continues to count after the preload. It is software's responsibility to ensure that the counter is in the desired state (example: execute stop command) prior to issuing a preload command. All others reserved.
15	0b	Reserved



Table 609. PMON Command Register 0-7 - PMON_CMD[0:7] (Sheet 4 of 4)

Register Offset																Attribute Legend:																	
PMON_CMD0	+000h															RV = Reserved																RW = Read/Write	
PMON_CMD1	+010h															PR = Preserved																RC = Read Clear	
PMON_CMD2	+020h															RS = Read/Set																RO = Read Only	
PMON_CMD3	+030h																															NA = Not Accessible	
PMON_CMD4	+040h																																
PMON_CMD5	+050h																																
PMON_CMD6	+060h																																
PMON_CMD7	+070h																																
Bit	Default	Description																															
14:12	000b	Command Trigger Source Select (CTSS) This field expands the Event Selection Codes in the Command trigger field to 1 of 8 ports, which share the same codes. Normally driven to 000, unless another source of the same type is to be selected. For 81341 and 81342, there are 8 different sources for SAS/SATA events, so setting these bits as "000" choose an event from SAS/SATA Port-0, and setting these bits as "001" choose an event from SAS/SATA Port-1, etc.																															
11:0	000h	Command Trigger (CT) This field contains the Event Selection Code (ESC) that the unit is required to detect before executing the opcode. The previously programmed opcode continues to execute until this command trigger is detected. The ESC of 000h (the default value) is a special case. This special ESC causes the command to be triggered immediately upon being written to the command register. The special command trigger also causes that command to be executed (triggered) once and only once. All other (non-0) ESCs cause the command to be re-executed every time the trigger is detected. Refer to the Event tables for valid values. For the Stop, Start, and Reset opcodes, re-executing every time the trigger is detected is meaningless. For other opcodes, care must be taken to prevent the CT from producing undesirable behavior. The only opcodes that are anticipated as being useful with continual triggering include the Restart functionality which behavior is necessary to facilitate gathering of histogram data at hardware speeds. Only one trigger can occur per core clock. When the trigger source clock is running faster than the core clock and more than one trigger event occurs per core clock, these additional triggers are dropped. See the Event Table for valid values.																															



18.5.6.2 PMON Event Register 0-7 - PMON_EVR[0:7]

This 32-bit register contains the events that control the incrementing and decrementing of the **PMON** counter selected. When this register is written, the previous register contents are overwritten (the event fields are unbuffered) and the associated counter is immediately affected by the change. This register should only be programmed when the counter is idle and before the command register receives an opcode that is associated with the events in this register. When both an increment and a decrement event are detected on the same clock cycle, the counter value does not change.

Table 610. PMON Event Register 0-7 - PMON_EVR[0:7]

		Register Offset																Attribute Legend:															
PMON_EVR0		+004h																RV = Reserved								RC = Read Clear							
PMON_EVR1		+014h																PR = Preserved								RO = Read Only							
PMON_EVR2		+024h																RS = Read/Set								NA = Not Accessible							
PMON_EVR3		+034h																															
PMON_EVR4		+044h																															
PMON_EVR5		+054h																															
PMON_EVR6		+064h																															
PMON_EVR7		+074h																															

Bit	Default	Description
31	0b	Decrement Occurrence Count Enable (DOCE) 0 = Decrement Duration Count: the counter is decremented for each clock for which the decrement event signal is asserted logic high 1 = Decrement Occurrence Count: the counter is decremented each time a rising edge of the decrement event signal is detected.
30:28	0h	Decrement Event Source Select (DESS) This field expands the Event Selection Codes in the Decrement Event field to 1 of 8 ports, which share the same codes. Normally driven to 000, unless another source of the same type is to be selected. For 81341 and 81342, there are 8 different sources for SAS/SATA events, so setting these bits as "000" choose an event from SAS/SATA Port-0, and setting these bits as "001" choose an event from SAS/SATA Port-1, etc.
27:16	000h	Decrement Event (DE) This field contains Event Selection Code (ESC) that the unit is required to detect before decrementing the associated counter. This field is only applicable for opcodes putting the counter in a counting state. Refer to the Event tables for valid values.
15	0b	Increment Occurrence Count Enable (IOCE) 0 = Increment Duration Count : the counter is incremented for each clock for which the increment event signal is asserted logic high 1 = Increment Occurrence Count : the counter is incremented each time a rising edge of the increment event signal is detected.
14:12	000b	Increment Event Source Select (IESS) This field expands the Event Selection Codes in the Increment Event field to 1 of 8 ports, which share the same codes. Normally driven to 000, unless another source of the same type is to be selected. For 81341 and 81342, there are 8 different sources for SAS/SATA events, so setting these bits as "000" choose an event from SAS/SATA Port-0, and setting these bits as "001" choose an event from SAS/SATA Port-1, etc.
11:0	000h	Increment Event (IE) This field contains the Event Selection Code (ESC) that the unit is required to detect before incrementing the associated counter. Refer to the Event tables for valid values.



18.5.6.3 PMON Status Register 0-7 - PMON_STS[0:7]

This 32-bit register reports the current status of the **PMONx** counter.

Table 611. PMON Status Register 0-7 - PMON_STS[0:7] (Sheet 1 of 2)

IOP Attributes PCI Attributes																																											
		Register Offset																Attribute Legend:																SC = Read Set/Write Clear									
PMON_STS0		+008h																RV = Reserved																RC = Read Clear									
PMON_STS1		+018h																PR = Preserved																RO = Read Only									
PMON_STS2		+028h																RS = Read/Set																NA = Not Accessible									
PMON_STS3		+038h																RW = Read/Write																									
PMON_STS4		+048h																																									
PMON_STS5		+058h																																									
PMON_STS6		+068h																																									
PMON_STS7		+078h																																									
Bit	Default	Description																																									
31	0b	Decrement Event Indicator (IEI) 0 = The selected decrement event was NOT detected. 1 = 1 = The selected decrement event WAS detected. This bit is updated every clock, whether or not the counter is counting.																																									
30	0b	Increment Event Indicator (IEI) 0 = The selected increment event was NOT detected. 1 = 1 = The selected increment event WAS detected. This bit is updated every clock, whether or not the counter is counting.																																									
29	0b	Counter Active Indicator (CAI) 0 = The associated counter is in a state that does NOT allow it to be incremented or decremented when the appropriate event(s) are detected. 1 = The associated counter is in a state that allows it to be incremented or decremented when the appropriate event(s) are detected. This bit is updated every clock.																																									
28	0b	In Use Software Mutex bit - After a full component RESET, a read to this bit returns a 0 - After the first read, subsequent reads return a 1 - A write of a 1 to this bit resets the next read value to 0 - Writing a 0 to this bit has no effect Software can poll this bit until it reads a 0, and then owns the usage of the corresponding counter. This bit has no other effect on any PMON Counter registers, and is only used as a semaphore among various independent software threads that may need to utilize this performance counter. Software that reads this register, but does not intend to claim exclusive access, must write a 1 to this bit (when it reads a 0), in order to allow other software threads to claim it. This bit is particularly useful when supporting multi-threaded environments. Attribute is Read Set/Write Clear (RS/WC)																																									
27	0b	Unsupported Event Indicator (UEI) 0 = No unsupported events have been selected. 1 = An unsupported Event Selection Code (ESC) was written into in one of the event selection fields (Command Trigger, Increment Event, or Decrement Event).																																									
26	0b	Overflow/Underflow Indicator (OUI) 0 = The associated 32 bit counter has NOT rolled over since the last time it was cleared. 1 = The associated 32 bit counter HAS rolled over since the last time it was cleared.																																									



Table 611. PMON Status Register 0-7 - PMON_STS[0:7] (Sheet 2 of 2)

Register Offset																Attribute Legend:																SC = Read Set/Write Clear					
PMON_STS0	+008h															RV = Reserved																RC = Read Clear					
PMON_STS1	+018h															PR = Preserved																RO = Read Only					
PMON_STS2	+028h															RS = Read/Set																NA = Not Accessible					
PMON_STS3	+038h															RW = Read/Write																					
PMON_STS4	+048h																																				
PMON_STS5	+058h																																				
PMON_STS6	+068h																																				
PMON_STS7	+078h																																				

Bit	Default	Description
25	0b	Command Trigger Indicator (CTI) 0 = NO commands have been triggered since the last time this bit was cleared. 1 = A command WAS triggered since the last time this bit was cleared. Software can use this bit to know that a command that was pending earlier has been triggered. Once a command has been triggered, another command can be triggered to execute. The null event does not cause this Command Trigger Indicator to be asserted.
24	0b	Threshold Indicator (THI) 0 = No threshold event has been generated since the last time this bit was cleared. 1 = This counter generated a threshold event due to a true threshold condition compare since the last time this bit was cleared.
23:12	000h	Reserved
11:4	TBD	Clock Period (CP) This fixed point field is 5.3 format which allows representing clock periods from 0.125 ns (8 GHz) to 31.875 ns (a little over 31 MHz) in 125 ps increments. Example: 100 MHz = 10.000 ns period = 01010.000b = 50h Example: 133 MHz = 7.500 ns period = 00111.100b = 3Ch Example: 167 MHz = 6.000 ns period = 00110.000b = 30h Example: 200 MHz = 5.000 ns period = 00101.000b = 28h
3:0	0h	Reserved



18.5.6.4 PMON Data Register 0-7 - PMON_DATA[0:7]

This 32-bit register allows for reading of the sampled value from **PMON** event counter X and contains the threshold value that is compared to the value in the event counter when a threshold condition (non-0 condition code) is in effect.

Table 612. PMON DATA Register 7-0 - PMON_DATA[7:0]

Register Offset		Attribute Legend: RW = Read/Write RV = Reserved RC = Read Clear PR = Preserved RO = Read Only RS = Read/Set NA = Not Accessible																											
PMON_DATA0	+00Ch																												
PMON_DATA1	+01Ch																												
PMON_DATA2	+02Ch																												
PMON_DATA3	+03Ch																												
PMON_DATA4	+04Ch																												
PMON_DATA5	+05Ch																												
PMON_DATA6	+06Ch																												
PMON_DATA7	+07Ch																												
Bit	Default	Description																											
31:0	0000 0000h	Counter X Value: Contains either duration (number of clock ticks) or occurrences contained in PMON event counter n at time of sampling. The register is programmed to contain the threshold value that is compared to the value in the event counter when non-0 condition codes have been selected.																											



18.5.7 PMON Events

PMON events can be selected for any of the counters. Each event is defined by a unique Event Selection Code (ESC) as defined below. For events which are duplicated within 81341 and 81342 with duplicate units (ADMA, SAS ports, etc.) source select field is used to select the unit the corresponding event is associated with. Event types are designated by Type, Occurrence only (O), Duration only (D) or events that can be selected as either (OD). Events must be used according to the defined type for proper operation. Use of a Duration only event, as an Occurrence event, results in unpredictable behavior.

Table 613. Event Selection Code Summary

ESC Range	Description
000-0FF	PMON Counters Internal Events • 000 = Null (False) Event • 001 - 00F = Clock Domains • 010 - 020 = Threshold Events • 021 - 0FF = Reserved for PMON internal events
100-3FF	Reserved
400-4FF	DDR-II SDRAM Memory Controller
500-5FF	Reserved
600-67F	Reserved
680-6FF	PCI-X ATU
700-7FF	PCI Express ATU
800-87F	North Internal XSI Bus
880-8FF	South Internal XSI Bus
900-FFF	Reserved

18.5.7.1 Null Event

The Null Event is not an actual event. When used as an increment or decrement event, no action takes place. When used as Command Trigger, it causes command to be triggered immediately after command register is written to by software. The Command Trigger Indicator status bit is NOT set when the Null Event is the Command Trigger. Also called False Event.

Table 614. Intel® 81341 and 81342 I/O Processors PMON Clock Events

Event Selection Code (Hex)	Event	Type	Comment
000	Null Event	N/A	Not an actual event



18.5.7.2 Clock Events

One of the clock events matches the frequency of the **PMON** unit clock. Software must comprehend that the frequency of this **PMON** unit clock may be different from one component to another. One piece of functionality this enables is to allow for event(s) to be counted during a fixed period of time. The Clock Event that matched the **PMON** Counter frequency can be used to trigger any command to execute. This essentially allows the command trigger capability to provide a programmable timer.

For example, when needing to know how many times event AAAh occurs during 1 millisecond, start counting event AAAh at the same time as when starting to count the **PMON** Counter frequency event. With a 100 MHz clock, event 001h would be started as an EQUAL Threshold Compare against the value 000186A0h (number of 100 MHz clock ticks in 1 ms). When the threshold compare of the counted clock events is true and therefore generates a threshold event, the threshold event triggers another counter to stop counting event AAAh.

Table 615. 81341 and 81342 PMON Clock Events

Event Selection Code (Hex)	Event	Type	Comment
001	XSI clock	D	up to 400MHz, SKU dependent
Others	Reserved		

18.5.7.3 Threshold Events

These are the outputs of the threshold comparators. When the value in a data register is compared to its corresponding counter value and the condition is true, a threshold event is generated. This results in:

- Pulse on the signal lines that are routed to the events input port (one signal line from each comparator).
- Pulse on the signal line that is routed to the **PMONOUT** output pin (one signal that is an OR of all signal lines from each comparator).

One piece of functionality this enables is to allow for **PMON** commands to be completed only when a Threshold Event occurs. In other words, a Threshold Event can be used as a Command Trigger to control the execution of any **PMON** command (start, stop, sample, etc.).

Table 616. 81341 and 81342 PMON Threshold Events

Event Selection Code (Hex)	Event	Type	Comment
010	Counter0 Threshold	OD	
011	Counter1 Threshold	OD	
012	Counter2 Threshold	OD	
013	Counter3 Threshold	OD	
014	Counter4 Threshold	OD	
015	Counter5 Threshold	OD	
016	Counter6 Threshold	OD	
017	Counter7 Threshold	OD	



18.5.7.4 DDR SDRAM Memory Controller Events

The DDR SDRAM Memory Controller Unit has multiple ports. Some events apply to each port and the following table represents the Source Select Field values for each port.

Table 617. DDR SDRAM Memory Controller Source Select Summary

Source Select Value	Port
0	XSI North Bus Interface
1	XSI South Bus Interface
2	ADMA 0 Port
3	ADMA 1 Port
4	ADMA 2 Port
5	LMU Prefetch
6-7	Reserved

Note: The ADMA is a requester on the South Internal Bus for PCI data transfers as well as descriptor fetches, descriptor status writes, CRC result word writes, XOR result writes and memory-to-memory DMA writes.



The events and corresponding codes for the DDR SDRAM Memory Controller are defined in [Table 618](#).

Table 618. DDR SDRAM Memory Controller Events (Sheet 1 of 3)

Event Selection Code (Hex)	Event	SRC	Type	Comment
40 1	DM Reads	N	O	DDR SDRAM Memory read transactions See events 480-4B5 for data count.
40 2	DM Writes	N	O	DDR SDRAM Memory write transactions See events 480-4B5 for data count.
40 3	DM Read Page Hit	N	O	DDR SDRAM Memory read page hit due to any memory access
40 4	DM Read Page Miss	N	O	DDR SDRAM Memory read page misses due to any memory access
40 5	DM Read Page Empty	N	O	DDR SDRAM Memory read page miss due to any memory access. There was no open page that had to be closed before the next read could occur.
40 6	DM Write Page Hit	N	O	DDR SDRAM Memory write page hit due to any memory access
40 7	DM Write Page Miss	N	O	DDR SDRAM Memory write page misses due to any memory access
40 8	DM Write Page Empty	N	O	DDR SDRAM Memory write page miss due to any memory access. There was no open page that had to be closed before the next read could occur.
40 9	reserved			
40 A	ECC Soft Error Corrections	N	O	of single bit DDR SDRAM Memory ECC errors corrected per 16 byte quanta. PMON
40 B	Waiting on Memory Refresh	N	OD	Reads or Writes waiting behind memory refresh
40 C-40 D	reserved			
40 E	DM Partial Writes	N	O	of partial writes to System Memory (sub 64-bit writes)
40 F-41 F	reserved			
42 0	DM Page 0 idle	N	OD	DDR SDRAM Memory Page 0 is idle (all pages closed)
42 1	DM Page 1 idle	N	OD	DDR SDRAM Memory Page 1 is idle (all pages closed)
42 2	DM Page 2 idle	N	OD	DDR SDRAM Memory Page 2 is idle (all pages closed)
42 3	DM Page 3 idle	N	OD	DDR SDRAM Memory Page 3 is idle (all pages closed)
42 4	DM Page 4 idle	N	OD	DDR SDRAM Memory Page 4 is idle (all pages closed)
42 5	DM Page 5 idle	N	OD	DDR SDRAM Memory Page 5 is idle (all pages closed)
42 6	DM Page 6 idle	N	OD	DDR SDRAM Memory Page 6 is idle (all pages closed)
42 7	DM Page 7 idle	N	OD	DDR SDRAM Memory Page 7 is idle (all pages closed)
42 8	DM Page 8 idle	N	OD	DDR SDRAM Memory Page 8 is idle (all pages closed)
42 9	DM Page 9 idle	N	OD	DDR SDRAM Memory Page 9 is idle (all pages closed)
42 A	DM Page 10 idle	N	OD	DDR SDRAM Memory Page 10 is idle (all pages closed)
42 B	DM Page 11 idle	N	OD	DDR SDRAM Memory Page 11 is idle (all pages closed)
42 C	DM Page 12 idle	N	OD	DDR SDRAM Memory Page 12 is idle (all pages closed)
42 D	DM Page 13 idle	N	OD	DDR SDRAM Memory Page 13 is idle (all pages closed)

**Table 618. DDR SDRAM Memory Controller Events (Sheet 2 of 3)**

Event Selection Code (Hex)	Event	SRC	Type	Comment
42E	DM Page 14 idle	N	OD	DDR SDRAM Memory Page 14 is idle (all pages closed)
42F	DM Page 15 idle	N	OD	DDR SDRAM Memory Page 15 is idle (all pages closed)
430	DM Page Switch	N	O	DDR SDRAM Memory controller switched one page to another.
431 - 47F	reserved			
480	DM Read Data	N	D	DDR SDRAM Memory read by memory controller (# DDR bursts: burst = 32bytes with 64-bit data bus width). Includes read for RMW of partial writes: See also ESC 40E.
481	DM Port Read Data	Y	D	Amount of DDR SDRAM data read from corresponding port. (# DDR bursts: burst = 32bytes with 64-bit data bus width).
482	Reserved			
483	DM Read Request Queue Enter	Y	O	Read Transaction entered DDR SDRAM Memory controller read request queue. 1 of 3 signals required to track the number of entries in this queue.
484	DM Read Request Queue Exit	Y ^a	O	Read Transaction exited DDR SDRAM Memory controller read request queue. 1 of 3 signals required to track the number of entries in this queue. 1 of 2 signals required to generate a Head of Queue Histogram for this queue.
485	DM Read Request Queue Empty	Y ^a	OD	DDR SDRAM Memory controller read request queue has no entries. 1 of 3 signals required to track the number of entries in this queue.
486	DM Read Request Queue Not Empty	Y ^a	OD	DDR SDRAM Memory controller read request queue has one or more entries. 1 of 2 signals required to generate a Head of Queue Histogram for this queue.
487	DM Read Request Queue Full	Y ^a	OD	DDR SDRAM Memory controller read request queue is completely full of transactions.
488	DM Write Data	N	D	DDR SDRAM Memory written by memory controller (DDR bursts: burst = 32bytes with 64-bit data bus width). Includes partial writes: See also ESC 40E.
489	DM Port Write Data	Y	D	Amount of DDR SDRAM data written from corresponding port. (# DDR bursts: burst = 32bytes with 64-bit data bus width).
48A	Reserved			
48B	DM Write Request Queue Enter	Y	O	Write Transaction entered DDR SDRAM Memory controller write request queue. 1 of 3 signals required to track the number of entries in this queue.
48C	DM Write Request Queue Exit	Y ^a	O	Write Transaction exited DDR SDRAM Memory controller write request queue. 1 of 3 signals required to track the number of entries in this queue. 1 of 2 signals required to generate a Head of Queue Histogram for this queue.
48D	DM Write Request Queue Empty	Y ^a	OD	DDR SDRAM Memory controller write request queue has no entries. 1 of 3 signals required to track the number of entries in this queue.
48E	DM Write Request Queue Not Empty	Y ^a	OD	DDR SDRAM Memory controller write request queue has one or more entries. 1 of 2 signals required to generate a Head of Queue Histogram for this queue.
48F	DM Write Request Queue Full	Y ^a	OD	DDR SDRAM Memory controller write request queue is completely full of transactions.

**Table 618. DDR SDRAM Memory Controller Events (Sheet 3 of 3)**

Event Selection Code (Hex)	Event	SRC	Type	Comment
490	DM Read Coherent Hits	Y	O	of times a read transaction has a coherency conflict with a write in a port transaction queue. (count is per hit, not per read transaction in the case of different write transactions with coherent hits with a single read transaction).
491	DM Write Coherent Hits	Y	O	of hits in this port's read request queue for coherency conflicts from a write transactions in the memory controller.
492	DM Transaction Count Expire	Y	O	of times this port's transaction count expires with additional transactions pending.
493 - 4FF	Reserved			

a. Source Selection valid for North and South Internal Busses only.



18.5.7.5 PCI Interface Events

The PCI Interface Events apply to the ATU-X unit.

Table 619. PCI Interface Events

Event Selection Code (Hex)	Event	SRC	Type	Comment
680	PCI Inbound Data Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes). Data transferred for all cases where I/O Processor is target of transactions. Sum of events 681 and 682.
681	PCI Inbound Rd Data Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes). Data transferred for an Inbound Read where I/O Processor is target of read. Read could be satisfied with immediate data in PCI mode or with a Split Completion in PCI-X
682	PCI Inbound Data Wr Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes)
683-687	reserved			
688	PCI Out bound Data Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes). Data transferred for all cases where I/O Processor is initiator of transactions. Sum of events 689 and 68A.
689	PCI Outbound Rd Data Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes). Data transferred for an Outbound Read where I/O Processor is initiator of the read. Read could be satisfied with immediate data in PCI mode or with a Split Completion in PCI-X
68A	PCI Outbound Data Wr Transferred	N	D	Count equals number of 8_byte data cycles (actual data transferred can be 1 to 8 bytes).
68B-68F	reserved			
690	PCI Inbound Memory Reads	N	O	
691	PCI Inbound Memory Writes	N	O	
692	PCI Inbound Split Completions	N	O	
693 - 695	Reserved			
696	PCI Inbound Config Reads	N	O	
697	PCI Inbound Config Writes	N	O	
598 - 6AF	Reserved			
6B0	PCI Outbound Memory Reads	N	O	
6B1	PCI Outbound Memory Writes	N	O	
6B2	PCI Outbound Split Completions	N	O	
6B3-6EF	reserved			
6F0	PCI Inbound Read Retries	N	O	Retry due to full queue
6F1	Reserved			
6F2	PCI Inbound Write Retries	N	O	Retry due to full queue
6F3 - 6FF	Reserved			



18.5.7.6 PCI Express* Interface Events

The PCI Interface Events apply to the ATU-E unit.

Table 620. PCI Express* Interface Summary

Event Selection Code	Event	SRC	Type	Comment
700-70F	Reserved			
710	Total TLPs Transmitted	N	O	Transaction Layer Packets
711	Total TLPs Received	N	O	The link layer put a TLP into the upstream transaction layer queues. This includes TLPs that fail CRC, or are malformed packets that eventually get dropped.
712	Total DLLPs Transmitted	N	O	Data Link Layer Packets Transmitted, including Flow Control Updates
713	Total DLLPs Received	N	O	Data Link Layer Packets Received including Flow Control Updates
714 - 71F	Reserved			
720	Inbound Data Transferred	N	OD	
721	Inbound Mem Read Request	N	O	Number of commands, not a amount of data
722	Inbound Mem Write Request	N	O	Number of commands, not a amount of data
723 - 73F	Reserved			
740	Outbound Data Transferred	N	OD	
741	Outbound Read Requests	N	O	Includes I/O, memory and config outbound reads
742	Outbound Write Requests	N	O	Includes I/O, memory and config outbound writes
743 - 74F	Reserved			
750	Correctable Error Message Received	N	O	
751	Non-Fatal Error Message Received	N	O	
752	Fatal Error Message Received	N	O	
753 - 78F	Reserved			
790	Correctable Error Detected	N	O	
791	Non-Fatal Error Detected	N	O	
792	Fatal Error Detected	N	O	
793 - 7FF	Reserved			



18.5.7.7 North Internal Bus Events

The North Internal Bus has multiple initiators. Some events apply to each requester unit and the following table represents the Source Select Field values for each unit.

Table 621. North Internal Bus Source Select Summary

Source Select Value	Port
0	Intel XScale® core 0
1	Intel XScale® core 1
2	Internal Bus Bridge
3:7	Reserved

The events and corresponding codes for the North Internal Bus are defined in the following table. These codes are unique to the IOP programming model of the **PMON** unit.

Table 622. North Internal Bus Initiator Events

Event Selection Code	Event	SRC	Type	Comment
800	NIB Addr Acq	Y	D	Address Acquisition Duration
801	NIB Addr Gnt	Y	O	Address Grants Received
802	NIB Data Acq	Y	D	Data Acquisition Duration
803	NIB Data Gnt	Y	O	Data Grants Received
804-807	Reserved			
808	NIB Snoop Retry	Y	O	# Transactions which receive a Snoop Retry
809	NIB Coherent Requests	Y	O	# Requests in Coherent Memory (XSI Bridge only)
80A-80F	Reserved			
810	NIB Reads	Y	O	# Read Transactions
811	NIB Read Data	Y	D	# Read Data Cycles (in 16-Bytes)
812 - 81F	Reserved			
820	NIB Writes	Y	O	# Write Transactions
821	NIB Write Data	Y	D	# Write Data Cycles (in 16-Bytes)
822 - 87F	Reserved			



18.5.7.8 South Internal Bus Events

The South Internal Bus has multiple initiators. Some events apply to each requester unit and the following table represents the Source Select Field values for each unit.

Table 623. South Internal Bus Source Select Summary

Source Select Value	Port
0	ATU-E
1	ATU-X
2	Internal Bus Bridge
3	Reserved
4	ADMA[2:0] ^a
5:7	Reserved

a. A breakdown of the individual ADMA channel activity can be found in the DDR SDRAM memory controller events

Note: The ADMA is a requester on the South Internal Bus for PCI data transfers as well as descriptor fetches, descriptor status writes, CRC result word writes, XOR result writes and memory-to-memory DMA writes.

The events and corresponding codes for the South Internal Bus are defined in the following table. These codes are unique to the IOP programming model of the **PMON** unit.

Table 624. South Internal Bus Initiator Events

Event Selection Code	Event	SRC	Type	Comment
880	SIB Addr Acq	Y	D	Address Acquisition Duration
881	SIB Addr Gnt	Y	O	Address Grants Received
882	SIB Data Acq	Y	D	Data Acquisition Duration
883	SIB Data Gnt	Y	O	Data Grants Received
884-887	Reserved			
888	SIB Snoop Retry	Y	O	# Transactions which receive a Snoop Retry
889	SIB Coherent Requests	Y	O	# Requests in Coherent Memory (Internal Bus Bridge only)
88A-88F	Reserved			
890	SIB Reads	Y	O	# Read Transactions
891	SIB Read Data	Y	D	# Read Data Cycles (in 16-Bytes)
892 - 89F	Reserved			
8A0	SIB Writes	Y	O	# Write Transactions
8A1	SIB Write Data	Y	D	# Write Data Cycles (in 16-Bytes)
8A2 - 8FF	Reserved			



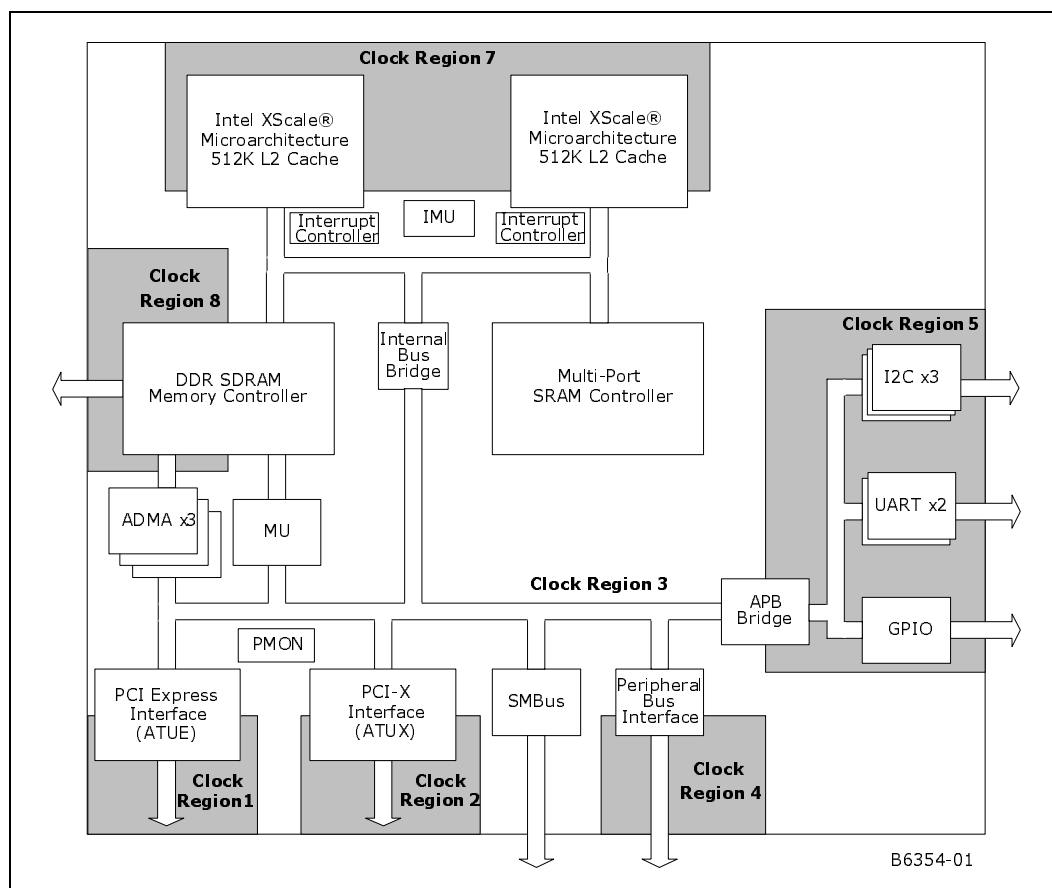
19.0 Clocking and Reset

This chapter describes the clocking and reset function of the Intel® 81341 and 81342 I/O Processors (81341 and 81342).

19.1 Clocking Overview

The 81341 and 81342 contains various internal clocking boundaries. Four PLLs are used to generate the clocks. One PLL for the PCI Express interface, one for PCI-X interface, one for Memory Interface and one for everything else. Clock regions 3, 4, and 6 are driven off the core PLL and are pseudo-synchronous to each other. There are asynchronous boundaries between regions 1/3, regions 2/3, 6/3, and between regions 7/3.

Figure 168. Intel® 81341 and 81342 I/O Processors Clocking Regions Diagram





19.1.1 Clocking Theory of Operation

Each region within the 81341 and 81342 contains different clocking requirements. These requirements are summarized in the following sections.

19.1.1.1 Clocking Region 1 (PCI Express)

Region 1 obtains its input clock from the PCI Express reference clock. The **REFCLK+/-** differential input supplies a 100 MHz clock for normal operation on the PCI Express interface. The analog front end for this region generates the 2.5 GHz clock used for the serial PCI Express interface as well as a 250 MHz clock used by the transaction layer. In addition to the locally generated clocks, region 1 utilizes the internal bus clock generated by the core PLL to interface to the internal bus (region 3).

Also contains a 25 MHz clock used by the power management logic.

19.1.1.2 Clocking Region 2 (PCI)

Region 2 obtains its input clock from the PCI bus clock connected to the input pin **P_CLKIN**. 81341 and 81342 supports an input frequency of 33 MHz, 66 MHz, 100 MHz, and 133 MHz for normal operation on the PCI interface. Additionally region 2 utilizes the internal bus clock generated by the core PLL to interface to the internal bus (region 3).



19.1.1.2.1 Central Resource Mode (PCIX_EP# = '1')

When operating as the Central Resource (**PCIX_EP#** = 1), the bus PCI bus operating frequency and default value in ATUX PCSR[19:16] is determined based on the sampling of **PCIXCAP**, **P_MODE2**, **M66EN**, **PCIXM1_100#**, and **PCIXM2_100#**.

Table 625. PCI Bus Frequency Initialization^a

P_PCIXCAP	P_MODE2	P_M66EN	PCIXM1_100#	PCIXM2_100#	PCI Bus Mode	PCI Bus Frequency	ATUX PCSR[19:16]
< 0.11VCC	— ^b	Ground	—	—	PCI	33 MHz	1111
< 0.11VCC	—	Not connected	—	—	PCI	66 MHz	1111
< 0.6VCC & > 0.11 VCC	GND	—	—	—	PCI-X Mode 1	66 MHz	1110
< 0.6VCC & > 0.11VCC	VCC	—	—	GND	PCI-X Mode 2	100 MHz (PCI-X 200 MHz)	0101
< 0.6VCC & > 0.11VCC	VCC	—	—	VCC	PCI-X Mode 2	133 MHz (PCI-X 266 MHz)	0100
< 0.89VCC & > 0.6VCC	—	—	—	—	PCI-X Mode 1	66 MHz	1110
> 0.89VCC	—	—	GND	—	PCI-X Mode 1	100 MHz	1101
> 0.89VCC	—	—	VCC	—	PCI-X Mode 1	133 MHz	1100

a. 81341 and 81342 does not support PCI-X 533Mhz.

b. A '—' in table indicates value is a 'don't care' for computing bus mode/frequency. All signals must still be pulled to a valid logic level.

When PCI Express* REFCLK is the primary clock source, the **CR_FREQ[1:0]** pins can be used to control the operating frequency of an external clock source for the PCI bus. The **CR_FREQ[1:0]** value can be changed by setting bits[19:16] in the ATUX PCSR register (offset x074).

Table 626. CR_FREQ[1:0] Encoding

ATUX PCSR[19:16] (PCIX Init Pattern)	ATUX PCSR[10] (P_M66EN)	CR_FREQ[1:0]	Bus Frequency	Bus Mode
1111	0	11	33 MHz	Conventional PCI
1111	1	10	66 MHz	Conventional PCI
1110	—	10	66 MHz	PCI-X
1101	—	01	100 MHz	PCI-X
1100	—	00	133 MHz	PCI-X



19.1.1.2.2 cPCI Hot-Swap Mode (PCIX_EP# = '0' and HS_SM# = '0')

When operating in a Compact PCI Hot-Swap environment (**HS_SM#**=0, **PCIX_EP#**=0), the **HS_FREQ[1:0]** and **P_M66EN** pins are used to determine PCI Bus operating frequency and set default value in ATUX PCSR[19:16]. Inputs are sampled at trailing edge of reset and used to set dividers on region 2 PLL.

Table 627. HS_FREQ Encoding^a

HS_FREQ[1:0]	P_M66EN	Operating Mode	Bus Frequency	PCSR[19:16]
11	0	PCI	33 MHz	1111
11	1	PCI	66 MHz	1111
10	–	PCI-X (Mode 1)	66 MHz	1110
01	–	PCI-X (Mode 1)	100 MHz	1101
00	–	PCI-X (Mode 1)	133 MHz	1100

a. Hot-Swap is not supported in PCI-X Mode 2.

19.1.1.2.3 End Point Mode (PCIX_EP# = 0 and HS_SM# = 1)

When operating in end point mode, the setting of PCSR[19:16] is determined by the initialization pattern sampled off the PCI bus.

Table 628. PCI-X Initialization Pattern¹

PERR#	DEVSEL#	STOP#	TRDY#	ATUX PCSR[19:16]	Mode	Clock Frequency (MHz)	
						Minimum	Maximum
Deasserted	Deasserted	Deasserted	Deasserted	1111	PCI 33	16	33
				1111	PCI 66	33	66
Deasserted	Deasserted	Deasserted	Asserted	1110	PCI-X 66 Mode 1	50	66
Deasserted	Deasserted	Asserted	Deasserted	1101	PCI-X100 Mode 1	66	100
Deasserted	Deasserted	Asserted	Asserted	1100	PCI-X 133 Mode 1	100	133
Deasserted	Asserted	Deasserted	Deasserted	Reserved	PCI-X Mode 1	Reserved	
Deasserted	Asserted	Deasserted	Asserted		PCI-X Mode 1		
Deasserted	Asserted	Asserted	Deasserted		PCI-X Mode 1		
Deasserted	Asserted	Asserted	Asserted		PCI-X Mode 1		
Asserted	Deasserted	Deasserted	Deasserted		PCI-X (Mode 2)		
Asserted	Deasserted	Deasserted	Asserted	0110	PCI-X 66 (Mode 2)	50	66
Asserted	Deasserted	Asserted	Deasserted	0101	PCI-X 100 (Mode 2)	66	100
Asserted	Deasserted	Asserted	Asserted	0100	PCI-X 133 (Mode 2)	100	133
Asserted	Asserted	Deasserted	Deasserted	Reserved	PCI-X	Reserved	
Asserted	Asserted	Deasserted	Asserted		PCI-X		
Asserted	Asserted	Asserted	Deasserted		PCI-X		
Asserted	Asserted	Asserted	Asserted		PCI-X		

1. 81341 and 81342 supports neither PCI-X 533 Mode nor ECC in Mode 1.



19.1.1.2.4 Secondary Clock Outputs

This component has the ability to provide four PCI bus clocks (**P_CLKO[3:0]**) to drive external components as well as a dedicated feedback clock (**P_CLKOUT**) to drive the ATUX PCI interface. These clock outputs are can only be used when the PCI Express* reference clock (**REFCLK+/-**) is used as the primary chip clock and the ATUX is enabled and configured to operate as a Central Resource.

Table 629. Secondary Clock Output Control

PCIX_EP#	CLK_SRC_PCIE#	INTERFACE_SEL_PCIX#	PCIX/PCIe Interfaces Active	Clock Outputs Enabled
0 (End Point)	–	–	–	No
1 (Central Resource)	1 (PCLKIN)	–	–	No
1 (Central Resource)	0 (REFCLK+/-)	1 (PCIe)	–	No
1 (Central Resource)	0 (REFCLK+/-)	0 (PCIX)	–	Yes
1 (Central Resource)	0 (REFCLK+/-)	–	Dual Interface	Yes



19.1.1.3 Clocking Region 3 (Internal Bus)

Region 3 covers the 81341 and 81342 internal bus and obtains its input clock from either **REFCLK+/-or P_CLKIN** depending on the setting of the **CLK_SRC_PCIE#** strap. Region 3 operates at a frequency up to 400 MHz. All units interface to or reside in this region. The units which interface to this region include the Intel XScale® processors, the Memory Controller Unit, ATU and bridge to the AHB bus. Units which wholly reside within region 3 include the Application DMA, XSI Bridge, SMBus interface, and Peripheral Bus interface.

Region 3 contains an open-drain bi-directional clock (SMBCLK) used by the SMBus interface. The SMBCLK operates at a maximum clock frequency of 100 KHz.

19.1.1.4 Clocking Region 4 (Peripheral Bus Interface)

Region 4 obtains its input clock from the clocking unit specified in region 3. This region operates at a fixed 66 MHz and depending on the frequency of region 3 may be an asynchronous boundary.

19.1.1.5 Clocking Region 5

Region 5 obtains its input clock from the clocking unit specified in region 3. This region is used for low-speed peripheral units. Currently, this includes:

- I²C bus interface
- General-Purpose I/O unit
- UART serial bus interface

Region 5 contains an output clock (SCL) used for the I²C bus interface. The SCL clock frequency is 100 KHz or 400 KHz. SCL is generated from the internal bus clock with its frequency determined by the Fast Mode bit in the [I2C Control Register x — ICRx](#). The UART input clock is driven at 16.67 MHz, and divided within the unit for the serial interface baud rate.

19.1.1.6 Clocking Region 7 (Intel XScale® Processor)

Region 7 obtains its input clock from either **REFCLK+/-or P_CLKIN** depending on the setting of the **CLK_SRC_PCIE#** strap. This region is the Intel XScale® processor (ARM* architecture compliant). It supports clock frequencies up to a maximum of 1200 MHz operation. The region 7 clock is an integer multiple of the Internal Bus clock (region 3).

19.1.1.7 Clocking Region 8 (DDR SDRAM)

Region 8 provides six DDR SDRAM output clocks, and the clocking for the Memory Controller Unit. The clocking unit contains three output clocks, called **M_CK[2:0]**, and three complement clock outputs called **M_CK[2:0]#**. The **M_CK[2:0]** and **M_CK[2:0]#** outputs are used by the DDR SDRAM memory subsystem. It supports clock frequencies up to a maximum of 533 MHz operation for DDR533 SDRAM

The MCU operates at 1/2 the DDR frequency. A 2x clock is also required to drive all the request strobes.

MCU	DDR	2x Clock
200	400	400
267	533	533



19.1.2 Clocking Region Summary

Table 630 summarizes all of the input clock pins, output clock pins, and clock strapping option pins used in 81341 and 81342.

Table 630. Clock Pin Summary (Sheet 1 of 2)

Pin	Input/Output	Description
P_CLKIN	Input	PCI Bus Input Clock: Provides timing for all PCI transactions
REFCLKP REFCLKN	Input	PCI Express Input Clock: Differential 100 MHz Clock
P_M66EN	Input	PCI 66 MHz enable.
TCK	Input	Test Clock: provides clock input for IEEE 1149.1 Boundary Scan Testing (JTAG).
HS_FREQ[1:0]/ CR_FREQ[1:0]	Strap/Output	Hot-Swap Frequency: While in Hot-Swap mode (HS_SM# = 0 and PCIX_EP# = 0), these pins are inputs which determine the PCI bus mode and frequency. See Section 19.1.1.2, "Clocking Region 2 (PCI)" on page 986 for more details. Central Resource Frequency: While operating in Central Resource Mode (PCIX_EP# = 1), these pins are outputs which control an external PCI-X clock generator.
P_CLKOUT	Output	PCI Bus Output Clock: When REFCLKN/REFCLKP are used, the I/O Processor can generate the PCI output clocks. This pin would then be connected to P_CLKIN and trace length matched to P_CLKO[3:0]. The P_CLKOUT and P_CLKO[3:0] outputs are enabled when the PCI-X Interface is operating as a central resource (PCIX_EP# = 1) and the PCI Express* input clock is used as the primary clock input (CLK_SRC_PCIE# = 0). See Section 19.1.1.2.4, "Secondary Clock Outputs" on page 989 for more details.
P_CLKO[3:0]	Output	PCI Bus Output Clocks: When REFCLKN/REFCLKP are used, the I/O processor can generate the PCI output clocks. These pins then provide the PCI clocks to devices on the PCI bus. The P_CLKOUT and P_CLKO[3:0] outputs are enabled when the PCI-X Interface is operating as a central resource (PCIX_EP# = 1) and the PCI Express* input clock is used as the primary clock input (CLK_SRC_PCIE# = 0). See Section 19.1.1.2.4, "Secondary Clock Outputs" on page 989 for more details.
M_CK[2:0] M_CK#[2:0]	Output	Memory Clocks: Three differential clock pairs for unbuffered DIMMs for the external SDRAM memory subsystem. Registered DIMMs only use the M_CK[0]/M_CK#[0] pair.
CKE[1:0]	Output	SDRAM Clock Enable: Enables the clocks for the SDRAM memory. Deasserting places the SDRAM in self-refresh mode.
SCL[2:0]	Input/Output	I²C Clock: These are bi-directional Open Drain clocks that provides for synchronous operation of the I ² C buses.
SMBCLK	Input/Output	SMBus Output Clock: Bi-directional open-drain clock that provides for synchronous operation of the SMBus.
CLK_SRC_PCIE#	Strap	Selects the clock source used to drive the internal logic and Intel XScale® processor. 0 = Source clock is the PCI Express* reference clock (REFCLKN/REFCLKP). Note that if this selection is made, the REFCLKN/REFCLKP must be provided regardless of the PCI Express interface mode: Endpoint or Root Complex. 1 = Source clock is the PCI input clock (P_CLKIN). Note that if this selection is made, the P_CLKIN must be provided regardless of the PCI-X interface mode: Endpoint or Central Resource. Note: CLK_SRC_PCIE# is not sampled during a WARM Reset.



Table 630. Clock Pin Summary (Sheet 2 of 2)

Pin	Input/Output	Description
MEM_FREQ[1:0]	Strap	Memory Frequency: Determines the frequency of the DDR2 SDRAM memory subsystem. MEM_FREQ[1:0] 00 Reserved 01 Reserved 10 DDR-II SDRAM @ 533 MHz 11 DDR-II SDRAM @ 400 MHz (default)
PCIXM1_100#	Strap	PCI-X Mode 1 100 MHz Enable: When operating as the Central Resource (PCIX_EP# = 1) this strap limits the PCI-X bus to 100 MHz when operating in Mode 1.
PCIXM2_100#	Strap	PCI-X Mode 2 100 MHz Enable: When operating as the Central Resource (PCIX_EP# = 1) this strap limits the PCI-X bus to 100 MHz when operating in Mode 2.



19.2 Reset Overview

19.2.1 Fundamental Reset

There are four fundamental (hardware) resets for the 81341 and 81342. The main power on reset is controlled through the PCI reset signal (**P_RST#**). When this signal is asserted, the entire I/O Processor is placed in a reset state. The other resets have differing behavior based on strapping options and chip mode. The reset straps are sampled at the exit of all fundamental resets.

- **P_RST#** — This is an asynchronous input pin which resets the entire chip. All reset straps are sampled at the rising edge of P_RST#.
- **WARM_RST#** — This is an asynchronous input pin which resets the entire chip with the exception of 'sticky' bits in the PMMR registers. All reset straps are sampled at the rising edge of WARM_RST#.
- **PCI Express Hot Reset** — When operating as an endpoint and the Hot Reset sequence is received in the TS1 ordered set, the entire chip is reset with the exception of the PCI Express physical layer and 'sticky' bits in the PMMR registers. This reset is not applicable when the PCI Express interface is disabled. A subset of straps are sampled as described in [Section 19.5, "Reset Strapping Options"](#)
- **PCI Express Loopback** — When operating as an endpoint and the Loopback sequence is received in the TS1 ordered set, the entire chip is reset with the exception of the PCI Express physical layer and "sticky" bits in the PMMR registers. This reset is not applicable when the PCI Express interface is disabled. A subset of straps are sampled as described in [Section 19.5, "Reset Strapping Options"](#)
- **PCI Express Disable Link** — When operating as an endpoint and the Disable Link sequence is received in the TS1 ordered set, the entire chip is reset with the exception of the PCI Express physical layer and "sticky" bits in the PMMR registers. This reset is not applicable when the PCI Express interface is disabled. A subset of straps are sampled as described in [Section 19.5, "Reset Strapping Options"](#)
- **PCI Express Link Down** — When operating as an endpoint and the PCI Express* Link transitions to the link down state, the entire chip, including the PCI Express* physically layer, is reset with the exception of "sticky" bits in the PMMR registers. This reset is not applicable when the PCI Express* interface is disabled. A subset of straps are sampled as described in [Section 19.5, "Reset Strapping Options"](#)

19.2.2 Software Reset

In addition to the fundamental resets, 81341 and 81342 provides software control to reset the internal bus, and Intel XScale® processor. Reset straps are not re-sampled due to these resets.

- **Internal Bus Reset Bit** — This reset can be initiated in two ways. The first is by writing to the coordinated reset bits in the MU [Section 4.9.4, “Inbound Interrupt Status Register - IISR” on page 429](#). The second is via the watchdog timer as described in [Section 12.1.2, “Watch Dog Timer Operation” on page 809](#). This reset is specific to the integrated I/O processor and the associated peripheral units. PCI Configuration Registers are preserved through this reset. See [Section 19.2.8, “Internal Bus Reset”](#) for more details.
- **Intel XScale® Processor Reset Bit** — This reset is initiated through the **HOLD_X0_IN_RST# /HOLD_X1_IN_RST#** straps, the ATUX [Section 2.13.41, “PCI Configuration and Status Register - PCSR” on page 183](#), or the ATUE [Section 3.16.41, “PCI Configuration and Status Register - PCSR” on page 331](#). Once invoked, the Intel XScale® processors are held in reset until released by software. See [Section 19.2.7, “Intel XScale® Processor Reset Mechanism”](#) for more details.
- **Targeted Core Reset** — The Targeted Core reset can be used by an Intel XScale® processor to imitate a reset to another core in the system, including itself.

19.2.3 Secondary Bus Reset

When operating as a root complex or central resource, the following ‘secondary bus’ resets apply. Reset straps are not re-sampled due to these resets.

- **PCI Express Hot Reset** — When operating as a root complex, the ATUE can generate the PCI Express Hot Reset sequence in order to reset the downstream components on the PCI Express interface. This is accomplished by setting bit 0 of the [“PCI Express Link Control/Status Register - PELCSR” on page 338](#) in the ATUE. ([Section 3.16.45](#))
- **PCI Bus Reset** — When operating as the central resource on the PCI/X bus, the P_RSTOUT# output can be used to reset the downstream PCI bus. This is accomplished by writing to bit 21 of the [“PCI Configuration and Status Register - PCSR” on page 183](#) in the ATUX ([Section 2.13.41](#)). Bit 21 of PCSR defaults to a ‘1’. It is the responsibility of the firmware to clear the Central Resource PCI Bus Reset bit. After firmware clears bit 21 of PCSR hardware keeps the P_RSTOUT# signal asserted (low) for about 300uS - the hardware waits about 150uS to allow the PLL to warm-up and another 150uS to allow the clocks to stabilize. Therefore, firmware has to wait about 300 uS after clearing bit 21 of PCSR. After the 300uS has elapsed, hardware de-asserts the P_RSTOUT# signal. After P_RSTOUT# de-asserts, firmware has to wait before issuing the first configuration cycle in order to meet the PCI timing parameter Trhfa (about 2²⁶ PCI clocks). Note that the PCI timing parameter Trhfa is dependent on the PCI bus speed selected.



19.2.4 PCI Reset

This is the primary reset input for both the PCI Express and PCI/X interfaces. The **P_RST#** reset clears all internal state machines and logic, and initialize all registers, including sticky bits, to their default states. The assertion and deassertion of the PCI reset signal **P_RST#** is asynchronous with respect to **P_CLKIN/REFCLK+/-**. The rising edge of the **P_RST#** signal must be monotonic through the input switching range and must meet the minimum slew rate. The PCI local bus specification defines the assertion of **P_RST#** for a period of 1 ms after power is stable.

Upon the assertion of **P_RST#**, all units within the component are reset.

Upon the deassertion of **P_RST#**, the strapping pins are sampled to set configuration modes (refer to [Section 19.5, "Reset Strapping Options" on page 1002](#)).

19.2.5 PCI Express* Hot Reset

The PCI Express* specification defines an in-band reset sequence that is used to reset the link and downstream components. The Root Complex communicates the fact that it is entering and coming out of a reset using these messages and the downstream devices respond by also going through a reset. This incoming message by nature of the PCI Express* protocol is asynchronous to the reference clock.

As an End Point, the PCI Express Hot Reset clears all internal state machines and logic, and initialize all registers to their default states except 'sticky' error bits which are persistent through this reset.

As a root complex, the 81341 and 81342 can generate the reset message. In this case, the Hot Reset does not cause a full chip reset and does not affect the internal logic.

19.2.6 WARM_RST# Reset Mechanism

The **WARM_RST#** reset clears all internal state machines and logic, and initialize all registers to their default states except 'sticky' error bits which are persistent through reset. To eliminate potential system reliability problems, all devices are also required to either tristate their outputs or to drive them to safe levels during such a power on reset.



19.2.7 Intel XScale® Processor Reset Mechanism

This reset is initiated through:

- the **HOLD_X0_IN_RST#** / **HOLD_X1_IN_RST#** straps

Once invoked, the Intel XScale® processors are held in reset until released by software

The **HOLD_X0_IN_RST#** / **HOLD_X1_IN_RST#** are sampled at the trailing edge of the fundamental resets and control the default value of the Core Processor Reset bits in function 0. When invoked via the strap, software should clear the Core Processor Reset bits in function 0 to bring the Intel XScale® processor out of reset.

After the initial boot sequence, the host driver can place the Intel XScale® processor in reset by asserting the Initiate Core Reset bits in any of the enabled functions which causes the corresponding Core Processor Reset bits to be set. Software must clear the Core Processor Reset bits in the same function in order to bring the Intel XScale® processor out of reset.



19.2.8 Internal Bus Reset

This reset can be initiated through:

- The coordinated reset bits in the MU [Section 4.9.4, “Inbound Interrupt Status Register - IISR” on page 429](#).
- The watchdog timer as described in [Section 12.1.2, “Watch Dog Timer Operation” on page 809](#).

This function resets the Intel XScale® processor and all units on the internal bus, while preserving the PCI Configuration Registers.

Software must quiesce all PCI bus traffic before initiating the Internal Bus Reset.

1. Disable the ATU from either claiming or initiating new transactions by clearing the *Bus Master Enable* and the *Memory Enable* in the ATU Command Register.
2. Monitor the *Inbound Read Transaction Queue Status*, the *Outbound Read Transaction Queue Status*, and in PCI Express mode, the *Link Layer Retry Buffer Status* in the PCSR.
3. When the Inbound Read Transaction queue, the Outbound Read Transaction queue, and the Link Layer Retry Buffer are empty, software writes to the Coordinated Reset bit to initiate the Internal Bus Reset.

The Intel XScale® processor may or may not be held in reset, depending on the default value of the Core Processor Reset bit as described in [Section 19.2.7, “Intel XScale® Processor Reset Mechanism”](#).



When the reset internal bus bit in the PCI Configuration and Status Register is set, there are sideband signals notifying the ATUX and MCU that a reset is coming. Table 631, “Internal Bus Reset Summary” describes the operation of each unit:

Table 631. Internal Bus Reset Summary (Sheet 1 of 2)

Unit	Preparation for Reset	Reset Status
ATUX	The affect on the ATUX depends on the PCIX_EP# strap. End Point Mode (PCIX_EP# = 0): When the ATUX is informed that an internal bus reset is coming it does the following: PCI Interface Outbound Transaction: - When the ATUX has already asserted its PCI request signal, and not yet started a transaction, the ATUX deasserts its request and not start its transaction. - When the ATUX has not yet requested the PCI bus, the ATUX never asserts its request for the PCI bus. - When the ATUX is in the middle of a transaction, the ATUX performs the existing transaction. This means that an inbound write, the ATUX transfers as much data as available in the queue. When an outbound read, the ATUX reads the data until the transaction stops naturally (meaning that the target has ended the transaction or the ATUX has read all of the data it has been requested to read). Once terminated by the ATUX or the target, the ATUX no longer requests the PCI bus. - In PCI-X mode, the ATUX allows any outstanding split completions due to prior outstanding Split Requests to Master-Abort on the PCI Bus. Since, the IOP is only accessing Prefetchable Memory on the host, no error condition is created in the system. PCI Interface Inbound Transaction: - The ATUX no longer claims any new transactions on the PCI bus. This results in a master abort to the initiating master. - In PCI-X mode, data from an outstanding split request may not be returned to the host. It is the responsibility of the host software to handle this condition as a consequence of writing to the Internal Bus Reset bit. Internal Bus Interface: Inbound Transaction: - The ATUX goes ahead and assert their IB request signals, and try to continue any pending transactions as normal. There are no special actions taken on the internal bus for inbound transactions. Internal Bus Interface: Outbound Transaction: - For all ATUX outbound transactions, there are no special requirements since the internal units are reset. Upon meeting both the outbound and inbound transaction requirements, the ATUX asserts the sideband signal to the reset unit notifying it is ready-for-reset. Central Resource Mode (PCIX_EP# = 1) No special requirements, the ATUX can be reset at anytime. This reset includes the configuration space.	End Point Mode: Clear all ATUX queues and state machines. All ATUX Configuration Registers retain their current values. Central Resource Mode: Entire ATUX is reset, including Configuration Registers.

**Table 631. Internal Bus Reset Summary (Sheet 2 of 2)**

Unit	Preparation for Reset	Reset Status
ATUE	The ATUE does not participate in the IB Reset handshake. However the affect on the ATUE varies depending on the PCIE_RC# strap. End Point Mode (PCIE_RC# = 1): An Internal Bus Reset does not reset the ATUE when operating as an endpoint. Root Complex Mode (PCIE_RC# = 0) No special requirements, the ATUE can be reset at anytime. This reset includes the configuration space. Note: While the ATUX must take special requirements to prevent corrupting the PCI bus, the PCI Express link is a point-to-point interface so no precautions need to be taken before resetting the link.	End Point Mode: No affect on ATUE logic. All ATUX Configuration Registers retain their current values. Root Complex Mode: Entire ATUX is reset, including Configuration Registers. However, sticky bits in the Configuration Space are not reset.
MCU	MCU cannot be reset until the powerfail sequence completes.	Clear all interrupts and reset MMRs to default value
MU	MU can be reset at any time	PCI Configuration registers and MSI-X memory registers are preserved. All other MMRs and logic are reset to their default values.
All Other Units	No special requirements	Clear all interrupts and reset MMRs and logic to default value



19.3 Reset Pins

Table 632. Reset Pin Summary

Pin	Input/Output	Description
P_RST#	Input	Primary chip reset. Should be connected to PCI RST# or PCI Express PERST# depending on the mode of operation.
WARM_RST#	Input	Warm Reset is the same as a cold reset, except sticky configuration bits are not reset. This pin should only be used when the sticky bit functionality is required. In this scenario, the WARM_RST# pin must be tied to the system reset PCI_RST# signal while the P_RST# pin can be tied to the system power good signal. When the sticky bit functionality is not required, the WARM_RST# pin should not be used and must be tied to Vcc. When the PCI Express interface is used as an endpoint, the PCI Express inband Hot Reset Mechanism can also be used to provide the sticky bit functionality. Note: Driving WARM_RST# using any other methods than suggested above may result in unpredictable behavior of the device.
P_RSTOUT#	Output	Secondary Reset. This is used as the PCI bus reset when operating as the Central Resource
PB_RSTOUT#	Output	Peripheral Bus Reset: This signal is asserted whenever the internal logic is reset.
PCI Express Hot Reset	n/a	This is an inband reset message that can be received as an endpoint, and generated as a root complex.



19.4 Device Function Select

In all cases, the **INTERFACE_SEL_PCIX#** strap affects whether the part operates as a PCI Express or PCI-X device.



19.5 Reset Strapping Options

Table 633, “Reset Strap Signals” on page 1003 details the reset strapping options that are available to configure the component during reset. These straps are sampled and the component operating mode is determined at the deassertion of the fundamental reset. All the straps are sampled at the trailing edge of **P_RST#** and **WARM_RST#**; however, a subset of straps are sampled for other resets.

- **P_RST#** and **WARM_RST#**
Sample all straps at the deassertion of both **P_RST#** and **WARM_RST#**.
- PCI Express* Hot Reset, Loopback, Disable Link, and Link Down
When operating as a PCI Express* endpoint, the following straps are re-sampled at the deassertion of the reset condition.
 - CFG_CYCLE_EN#
 - HOLD_X0_IN_RST#
 - HOLD_X1_IN_RST#
- Software Reset (Internal Bus reset, Core reset, and so on)
Software Resets do not initiate a re-sampling of the reset straps and do not change the mode of operation of the component.
- Secondary Bus Reset (available when operating as Root Complex and/or Central Resource)
When operating as a PCI Express* Root Complex, the PCI Express* Hot Reset does not initiate a re-sampling of the reset straps.

When operating as a PCI-X Central Resource, P_RSTOUT# does not initiate a re-sampling of the reset straps.

**Table 633. Reset Strap Signals (Sheet 1 of 2)**

Name	Description
BOOT_WIDTH_8#	PBI Boot Bus Width: Indicates default bus width for the PBI Memory Boot window. 0 = 8 bits wide (Requires pull-down resistor) 1 = 16 bits wide (Default mode)
CFG_CYCLE_EN#	Configuration Cycle Enable: Determines when the PCI interface retries configuration cycles until. 0 = Configuration Cycles enabled (Requires pull-down resistor) 1 = Configuration Retry enabled (Default mode) <u>PCI-X Interface</u> Configuration cycles are claimed and terminated with a retry status. <u>PCI Express Interface</u> Configuration requests result in a completion TLP with Configuration Retry Status (CRS).
HOLD_X0_IN_RST#	Core 0 Processor Reset Mode: This strap is latched at the trailing edge of reset and reflected in Core 0 Processor Reset bit in function 0. See Section 19.2.7, "Intel XScale® Processor Reset Mechanism" for more details. When asserted, the associated Intel XScale® processor is held in reset until software clears the Core 0 Processor Reset bit. 0 = Hold in reset (Requires pull-down resistor) 1 = Don't hold in reset (Default mode).
HOLD_X1_IN_RST#	Core 1 Processor Reset Mode: This strap is latched at the trailing edge of reset and reflected in Core 1 Processor Reset bit in function 0. See Section 19.2.7, "Intel XScale® Processor Reset Mechanism" for more details. When asserted, the associated Intel XScale® processor is held in reset until software clears the Core 1 Processor Reset bit. 0 = Hold in reset (Requires pull-down resistor) 1 = Don't hold in reset (Default mode).
MEM_FREQ[1:0]	Memory Frequency: Determines the frequency of the DDR2 SDRAM memory subsystem. MEM_FREQ[1:0] 00 Reserved 01 Reserved 10 DDR-II SDRAM @ 533MHz 11 DDR-II SDRAM @ 400MHz (default)
INTERFACE_SEL_PCIX#	Selects the active interface and determines the address map for the PMMR registers. See the MMR chapter for details. 0 = PCI-X is active 1 = PCI Express is active (default mode) When both interfaces are active, this strap selects the ATU that is function 0 in the internal address map Note: For dual interface designs, INTERFACE_SEL_PCIX# must be set consistent with PCIE_RC# / PCIX_EP#. When operating with one interface as an endpoint and the other interface as a root complex, INTERFACE_SEL_PCIX# must correspond to the end point interface.
PCIE_RC#	PCI Express Root Complex: determines when the PCI Express interface operates as an endpoint or root complex. 0 = Root Complex (Requires pull-down resistor) 1 = Endpoint (Default Mode). Note: Setting both PCIE_RC# and PCIX_EP# to endpoint is unsupported and will result in unspecified operation.
PCIX_EP#	PCI-X End Point: determines when the PCI-X interface operates as an endpoint or central resource. 0 = Endpoint (Requires pull-down resistor) 1 = Central Resource (Default mode) Note: Setting both PCIE_RC# and PCIX_EP# to endpoint is unsupported and will result in unspecified operation.
PCIXM1_100#	PCI Bus Mode 1 100MHz Enable: limits the maximum PCI-X mode operating frequency to 100MHz while in mode1. Only used when ATU is acting as the central resource for the PCI domain. 0 = Limit maximum frequency to 100MHz.(Requires pull-down resistor) 1 = 133MHz enabled (Default mode)



Table 633. Reset Strap Signals (Sheet 2 of 2)

Name	Description
PCIXM2_100#	PCI-X Mode 2 133MHz Enable: limits the maximum PCI-X mode 2 operating frequency to 100MHz while operating in mode 2. 0 = Limit maximum Frequency to 100MHz (200MHz data). 1 = 133MHz (266MHz data) enabled (Default mode)
EXT_ARB#	External Arbiter: Determines whether the PCI interface enables the integrated arbiter, or uses an external arbiter. 0 = External Arbiter enabled (Requires pull-down resistor) 1 = Internal Arbiter enabled (Default mode)
PCIX_32BIT#	32-Bit PCI-X Bus: Sets the PCI-X bus width in the PCI-X Status Register. When this external strap is asserted, the Sunrise lake only uses the lower 32-bit of the data bus to operate and respond to PCI-X transactions. When the strap is deasserted the 81348 is able to operate and respond to both 32- and 64-bit PCI-X transactions. This external strap also controls how the REQ64# signal is driven when the ATU-X is used as central resource. As a central resource, REQ64# is driven accordingly based on the PCIX_32BIT# strap setting during the assertion of P_RSTOUT#. When PCIX_32BIT# is asserted REQ64# is driven high, and when PCIX_32BIT# is deasserted REQ64# is driven low. 0 = 32 Bit PCI-X Bus (Requires pull-down resistor) 1 = 64 Bit PCI-X Bus (Default mode)
HS_SM#	Hot-Swap Startup Mode 0 = Hot-Swap Mode Enabled (Requires pull-down resistor) 1 = Hot-Swap Mode Disabled (default mode)
SMB_A5 SMB_A3 SMB_A2 SMB_A1	SMBUS Address maps to address bits 5, 3, 2, and 1 of the SMBus Slave address. 0 = Address is low (Requires Pull-down resistor) 1 = Address is high (Default mode).
PCIX_PULLUP#	PCI-X Pull Up: determines when the PCI interface has On Die Pull Ups enabled. 0 = enable PCI pull up resistors 1 = disable PCI pull up resistors (Default mode).
FW_TIMER_OFF#	Firmware Timer Off: Disables the 400mS firmware timer. When enabled to timer automatically clears the Configuration Cycle Retry condition when the timer expires. 0 = firmware timer disabled 1 = firmware timer enabled (default mode)
CLK_SRC_PCIE#	Clock Source PCI-Express: selects the PCI-Express REFCLK pair as the input clock to the PLLs that control internal logic. 0 = source clock is REFCLKP/REFCKLN 1 = source clock is PCLK (default mode).
LK_DN_RST_BYPASS#	Link Down Reset Bypass: Disables the full chip reset that would normally be caused by a PCI Express Link Down or PCI Express Hot Reset. Refer to Section 19.2.1, "Fundamental Reset" on page 993 for reset descriptions. 0 = Do not reset on Link Down 1 = Reset on Link Down (default mode)



20.0 Test Logic Unit and Testability

20.1 Overview

This chapter summarizes testability and configuration features incorporated in the Intel® 81341 and 81342 I/O Processors (81341 and 81342).

The test and control logic is based on the IEEE 1149.1-2001 Standard Test Access Port and Boundary-Scan Architecture document (available from the IEEE). The TAP controller supports on-chip test logic such as Built-In Self Test and boundary-scan.

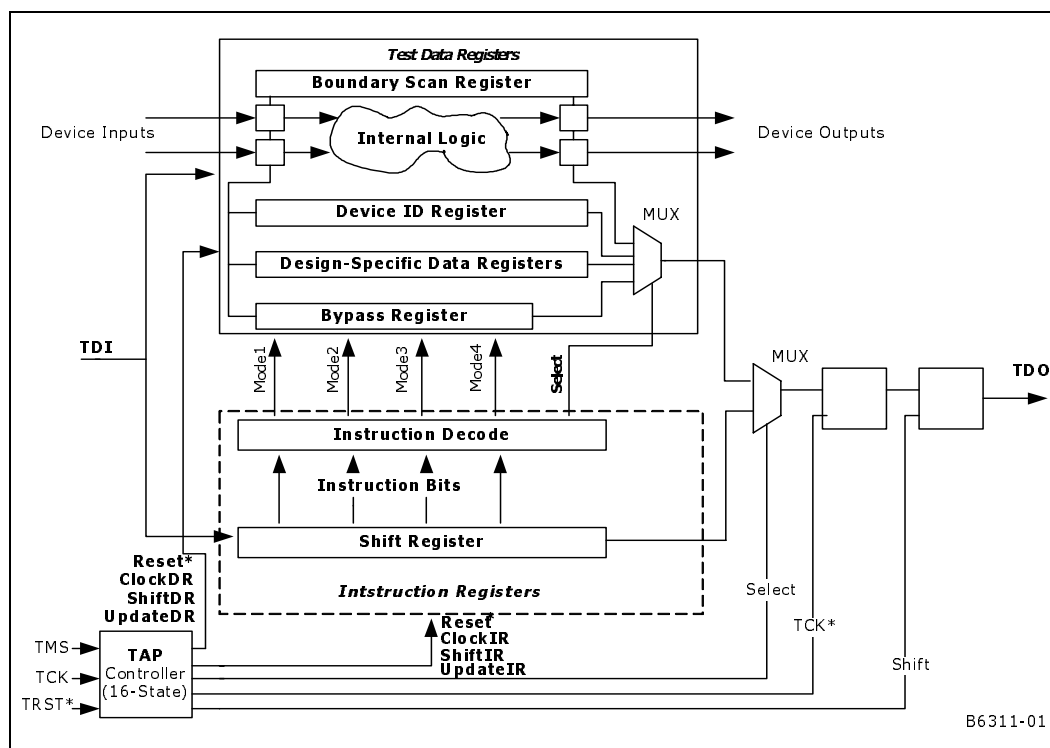
20.2 IEEE 1149.1 Standard Test Access Port (TAP)

The I/O processor contains test logic that is compatible with the IEEE Standard 1149.1-2001 Test Access Port (TAP) and Boundary Scan Architecture. Logic that conforms to this standard contains:

- Test Access Port (TAP):
 - four inputs (**TDI**, **TMS**, **TRST#** and **TCLK**)
 - single output (**TDO**).
- TAP controller
- Instruction register
- Group of test data registers

Each of these is described in more detail below. Figure 169 shows a generic diagram for logic conforming to the IEEE 1149.1 test standard.

Figure 169. IEEE 1149.1 Std. Block Diagram





20.2.1 TAP Pin Description

The internal test logic is accessed through the TAP pins. The following sections describe some of the rules and permissions of the IEEE 1149.1a Standard for the TAP pins.

20.2.1.1 Test Clock (TCK)

This is the clock input for the test logic defined by this standard, i.e. the TAP controller and associated registers. The TLU is a fully static design, thus all registers retain their states indefinitely when **TCK** is stopped at "0" or "1".

20.2.1.2 Test Mode Select (TMS)

This pin is used to control the operation of the TAP controller. The signal received at **TMS** is decoded by the TAP controller to control test operations. The state of **TMS** is sampled on the rising edge of **TCK**. Internally, there is a weak pull-up on this pin to provide a logic high when not driven, per standard definition.

20.2.1.3 Test Data Input (TDI)

This pin is used to provide serial input data to the instruction and test data registers. Data at **TDI** is sampled on the rising edge of **TCK**. Data shifted from **TDI** through a register to **TDO** appears non-inverted at **TDO** after a number of rising and falling edges of **TCK** determined by the length of the instruction or test data register selected. Internally, there is a weak pull-up on this pin to provide a logic high when not driven, per standard definition.

20.2.1.4 Test Data Output (TDO)

This is the serial data output pin. Changes in the state of **TDO** occur only following the falling edge of **TCK** while performing a shift operation. This pin is only driven while scanning (in SHDR or SHIR states) otherwise it is in inactive (high Z) state. The non-shift inactive state is provided to support parallel connection of **TDO** outputs at the board or module level.

20.2.1.5 Asynchronous Reset (TRST#)

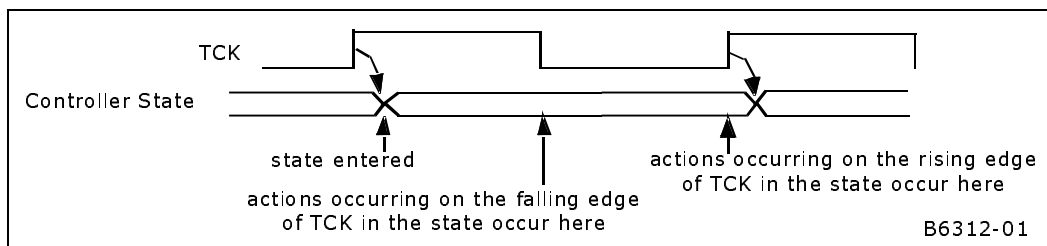
The **TRST#** signal is used to asynchronously reset the TAP controller and boundary-scan registers. The TAP controller is not initialized by any other system input, including system reset. The TAP controller initializes asynchronously on the falling edge of **TRST#** to the Test-Logic_Reset (initial) state. The TAP controller is initialized at power-up by circuitry built into the test logic. Upon reset, the TAP instruction register initializes to the IDCODE instruction. Internally, there is a weak pull-up on this pin to provide a logic high when not driven.

20.2.2 TAP Controller

The TAP controller, shown in Figure 171, is a sixteen-state synchronous finite state machine that changes state on the rising edge of **TCK**. The controller's next state is controlled by the state present at the **TMS** input. The TAP controller generates control signals, which together with **TCK** and control signals decoded from the instruction register, determine the operation of the test circuitry as defined by the IEEE Standard.

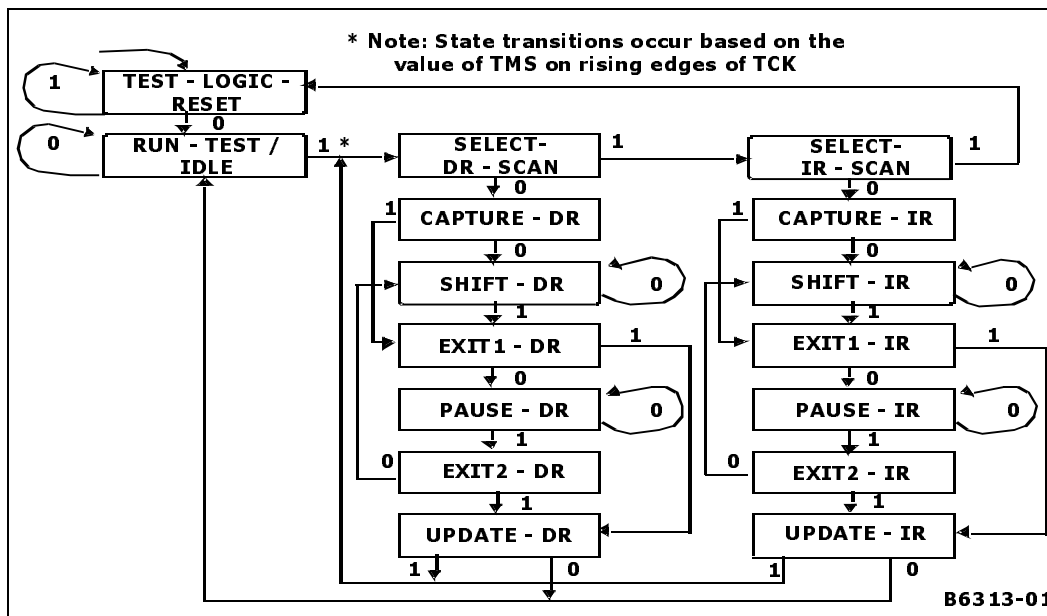
All state transitions occur based on values of **TMS** on the rising edge of **TCK**. Actions of the test logic (instruction register, data registers, etc.) occur on either rising or falling edge of **TCK**, as show in Figure 170.. See the description of each state to learn which.

Figure 170. Timing of Actions in a TAP Controller State



For greater detail on the state machine and the public instructions, refer to IEEE 1149.1a *Standard Test Access Port and Boundary-Scan Architecture Specification*.

Figure 171. TAP Controller State Diagram





20.2.2.1 Test-Logic-Reset State

In this state, test logic is disabled to allow normal operation of the Intel XScale® processor. This is achieved by loading the instruction register with the IDCODE instruction. No matter what the state of the controller, it enters Test-Logic-Reset state when the **TMS** input is held high for at least five rising edges of **TCK**. The controller remains in this state while **TMS** is high. The TAP controller is also forced to enter this state by enabling **TRST#**.

When the controller exits the Test-Logic-Reset controller state as a result of an erroneous low signal on the **TMS** line at the time of a rising edge on **TCK** (for example, a glitch due to external interference), it returns to the Test-Logic-Reset state following three rising edges of **TCK** with the **TMS** line at the intended high logic level. Test logic operation is such that no disturbance is caused to on-chip system logic operation as the result of such an error.

Transition to next state: On the rising edge of **TCK**, when **TMS** is low move to Run-Test/Idle, else **TMS** remains high so stay in Reset.

20.2.2.2 Run-Test/Idle State

This state is a controller state between scan operations. The controller remains in this state as long as **TMS** is held low. In the Run-Test/Idle state, activity in selected test logic occurs only when certain instructions are present. For example, the RUNBIST instruction causes on-chip self-tests to execute in this state. Instructions that do not cause functions to execute generate no activity in the test logic while the controller is in this state.

The instruction register and all test data registers retain their current value in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Select-DR-Scan, else remain in Idle.

20.2.2.3 Select-DR-Scan State

This is a temporary controller state. Here the decision is made to enter the Capture-DR column and initiate a scan sequence for the selected test data register.

All test data registers selected by the current instruction retain their previous value in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Capture-DR, else move to Select-IR.

20.2.2.4 Capture-DR State

When the controller is in this state data is parallel-loaded into test data registers selected by the current instruction on the rising edge of **TCK**. Test data registers that do not have parallel inputs are not changed. Also when capturing is not required for the selected instruction, the register retains its previous state.

The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Shift-DR, else move to Exit1-DR.



20.2.2.5 Shift-DR State

In this controller state, the test data register, which is connected between **TDI** and **TDO** as a result of the current instruction, shifts data one bit position nearer to its serial output on each rising edge of **TCK**. Test data registers that the current instruction select but do not place in the serial path, retain their previous value during this state.

The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit1-DR, else remain at Shift-DR.

20.2.2.6 Exit1-DR State

This is a temporary controller state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is held high on the rising edge of **TCK**, the controller enters the **Update-DR** state and the scanning process terminates. When **TMS** is held low on the rising edge of **TCK**, the controller enters the **Pause-DR** state.

20.2.2.7 Pause-DR State

The **Pause-DR** state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between **TDI** and **TDO**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to the Exit2-DR, else remain at Pause-DR.

20.2.2.8 Exit2-DR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, the controller enters the Update-DR state and the scanning process terminates. When **TMS** is held low on the rising edge of **TCK**, the controller re-enters the Shift-DR state.

20.2.2.9 Update-DR State

Data is latched into the parallel output of shift registers from the shift register path, on the falling edge of **TCK**.

All of the test data register's shift-register bit positions selected by the current instruction retain their previous values. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** remains high on the rising edge of **TCK**, then the controller moves to the Select-DR state, else the controller moves to the Run-Test/Idle state.



20.2.2.10 Select-IR-Scan State

This is a temporary controller state. Here the decision is made to enter the Capture-IR column and initiate a scan sequence for the instruction register or to return to Test-Logic-Reset.

All test data registers selected by the current instruction retain their previous value during this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Capture-IR, else move to Test-Logic-Reset.

20.2.2.11 Capture-IR State

In this state, the shift register contained in the instruction register loads the fixed value 0000001₂ on the rising edge of **TCK**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Shift-IR, else move to Exit1-IR.

20.2.2.12 Shift-IR State

In this state, the shift register contained in the instruction register is connected between **TDI** and **TDO** and shifts data one bit position nearer to its serial output on each rising edge of **TCK**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit1-IR, else remain at Shift-IR.

20.2.2.13 Exit1-IR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high during the next rising edge of **TCK**, the controller enters the Update-IR state and the scanning process terminates. When **TMS** is held low during the next rising edge of **TCK**, the controller enters the Pause-IR state.

20.2.2.14 Pause-IR State

This state allows the TAP controller to temporarily halt the shifting of data through the instruction register.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit2-IR, else remain in the Pause-IR state.



20.2.2.15 Exit2-IR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is held high during the next rising edge of **TCK**, the controller enters the Update-IR state and the scanning process terminates. When **TMS** is held low during the next rising edge of **TCK**, the controller re-enters the Shift-IR state.

20.2.2.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of **TCK**. Once latched, the new instruction becomes the current instruction.

All test data registers selected by the current instruction retain their previous values in this state.

Transition to next state: When **TMS** remains high on the rising edge of **TCK**, then the controller moves to the Select-DR state, else the controller moves to the Run-Test/Idle state.



20.2.3 TAP Controller Registers

The IEEE 1149.1 architecture specifies a minimum of two test data registers: bypass and boundary scan. The IOP TAP controller extends this number to allow access to test features within the device. Some registers are for public use, others are private. Per the standard, each test data register has a fixed length and can be accessed using one or more instructions. The 81348 design implements the following:

- Instruction Register
- Boundary Scan Register
- Bypass Register
- Device Identification Register

20.2.3.1 Instruction Register

Each of the TAP controllers in the design contain an instruction register (IR). Each IR is a 7-bit, master/slave-configured, parallel-loadable, serial-shift register with latched outputs. They are used in each unit to select the test data register to be accessed, the test to be performed or both.

When the TAP controller is in the **Shift-IR** state, instructions are loaded serially via **TDI** clocked by the rising edge of **TCK**. The shifted-in instruction becomes active upon latching from the serial-stages to the parallel-stages in the **Update-IR** state. At that time the IR outputs, along with the TAP finite state machine outputs, are decoded to select and control the test data register selected by that instruction. Upon latching, all actions caused by any previous instructions must terminate.

On activation of **TRST#**, the latched instruction asynchronously changes to the **IDCODE** instruction. Additionally, upon entering the **Test-Logic-Reset** state, the **IDCODE** instruction is latched and becomes active on the falling edge of **TCK**.

Because the TAP controllers are connected in series, each unit must receive its own 7-bit instruction during the Shift-IR state. The bypass instruction should be loaded into the units that are not to be accessed. When the Intel XScale® Processor Core 1 TAP controller is not MUXed into the TDI-TDO stream, the resulting instruction stream is 14-bits long. When the Intel XScale® Processor Core 1 TAP controller is inserted in the stream, the instruction stream is 21-bits long.

For example, to load the TEST_REG_READ instruction into the TLUs IR when the Intel XScale® Processor Core 1 TAP controller is not muxed into the TDI-TDO stream, the data shifted into TDI during the Shift-IR state is:

```

0   1   0   0   0   0   1   1   1   1   1   1   1   1   ----- >   TDI
TEST_REG_READ | BYPASS into
into TLU      | X-Scale core 0

```

As another example, to load the TEST_REG_READ instruction into the TLUS IR when the Intel XScale® Processor Core 1 TAP controller is muxed into the TDI-TDO stream, the data shifted into TDI during the Shift-IR state is:

0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	----->	TDI
TEST_REG_READ into TLU						BYPASS into X-Scale core 0						BYPASS into X-Scale core 1									



20.2.3.2 Instructions

Each of the TAP controller instruction sets is composed of both public and private instructions. Public instructions are intended for use by purchasers of the part. Since the instruction set for each TAP controllers is independent of one another, each is listed separately below

Table 634. TLU TAP Controller Instruction Set

Instruction	Opcode	Public/Private	Description
EXTTEST	0000000 ₂	Public	Intended for supporting the boundary-scan feature for testing device interconnects at the board/system level, the EXTTEST instruction connects only the boundary-scan register between TDI and TDO in the Shift-DR state. When EXTTEST is selected, all output signal pin values are driven by values shifted into the boundary-scan register and may change only on the falling-edge of TCK in the Update_DR state. Also, when exttest is selected, all system input pin states must be loaded into the boundary-scan register on the rising-edge of TCK in the Capture_DR state. Note: Data would typically be loaded onto the latched parallel outputs of boundary-scan shift registers using the SAMPLE/PRELOAD instruction prior to selection of the EXTTEST instruction.
SAMPLE/ PRELOAD	0000001 ₂	Public	SAMPLE/PRELOAD performs two functions: - When the TAP controller is in the Capture-DR state, the sample instruction occurs on the rising edge of TCK and provides a snapshot of the component's normal operation without interfering with that normal operation. The instruction causes Boundary-Scan register cells associated with outputs to sample the value being driven by or to the processor. - When the TAP controller is in the Update-DR state, the preload instruction occurs on the falling edge of TCK. This instruction causes the transfer of data held in the boundary-scan cells to the slave register cells. Typically the slave latched data is then applied to the system outputs by means of the extest instruction.
HIGHZ	0101110 ₂	Public	HIGHZ puts all output pins into a tri-state mode. When this instruction is active, the bypass register is connected between TDI and TDO.
CLAMP	0101111 ₂	Public	Once the clamp instruction is loaded into the TAP controller instruction register, the output pins are driven by the parallel output of the boundary-scan chain. The bypass register is selected as the serial path between TDI and TDO when the TAP controller passes through the Shift-DR state.
IDCODE	1111110 ₂	Public	IDCODE is used in conjunction with the device identification register. When selected, IDCODE parallel-loads the hard-wired identification code (32 bits) into the identification register on the rising edge of TCK following entry into the Capture-DR state. The instruction selects only the identification register for connection between TDI and TDO in the Shift-DR state for serial access. Note: The device identification register is not altered by data being shifted in on TDI.
BYPASS	1111111 ₂	Public	BYPASS selects the bypass register between TDI and TDO while in Shift-DR state, effectively bypassing the processor's test logic. 0 is captured in the Capture-DR state. While this instruction is in effect, all other test data registers have no effect on the operation of the system.



20.2.3.3 Boundary-Scan Register

Boundary-Scan Register is a set of serial-shiftable register cells, connected between each system pin and on-chip system logic. (Power, ground and TAP pins excluded.) This forms a single shift register between TDI and TDO of all the system pins.

This is the most extensive, complex register in the test circuitry. This register allows testing of circuitry external to the component (e.g. board interconnect) in addition to device system logic. It permits the system signals (into and out of the system logic) to be sampled and examined without causing interference with the normal operation of the system logic. Further definition, rules, and specifics of the Boundary-Scan Register can be found in the **IEEE Std, 1149.1-2001, Chapter 10**.

20.2.3.4 Bypass Register

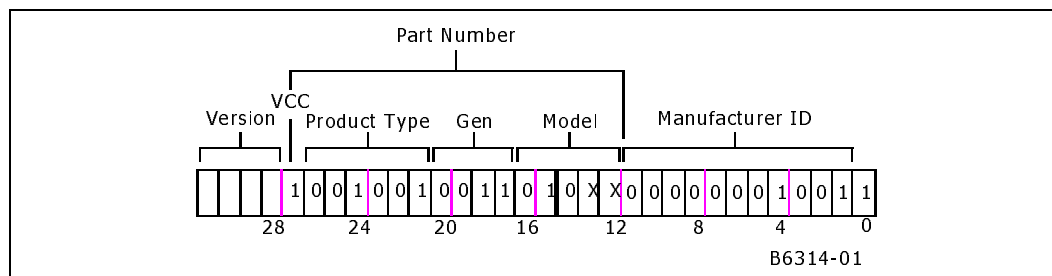
The Bypass Register is a single-bit, serial-shift register that connects **TDI** and **TDO** when the Bypass instruction is in effect. This allows rapid movement of test data to and from other components on the board, since this register provides the shortest path between **TDI** and **TDO**. This path can be selected when no test operation is being performed. While the Bypass Register is selected, data is transferred from **TDI** to **TDO** without inversion.

20.2.3.5 Device Identification Register

The Device Identification (ID) Register is a 32-bit register used for storing the manufacturer identification, part number, and the version of the processor. It is a dedicated part of the test logic and is not usable in system functionality.

The identification register is selected only by the **IDCODE** instruction. When the **Test-Logic-Reset** state of the TAP controller is entered, the IDCODE instruction is automatically loaded into the instruction register. The generic format of the register is discussed in chapter 11 of the IEEE 1149.1 Standard.

Figure 172. IOP Device ID Register



Note: See Intel® 81341 and 81342 I/O Processors Specification Update

Table 635. IOP Device ID Register Field Definitions

Field	Value	Definition
Version	0000 = A-0 Step	Indicates stepping changes.
Manufacturer ID	0000 0001 0011 (Indicates Intel)	Manufacturer ID assigned by IEEE.

Table 636. IOP Device ID Register Settings

Description	Device ID	
	Complete ID (Hex)	Complete ID (Binary)
A-0	TBD	TBD



20.3 Definition of Terms

High-Z:	An instruction defined by the IEEE 1149.1 Standard. The requirement for this instruction are 1) all system logic outputs are high impedance; 2) the TAP controller continues to operate with the bypass register connected between TDI and TDO. The part may have other settings, as long as they do not interfere with these two requirements. Following the use of High-Z the part may be in an indeterminate state and require a reset.
ONCE:	A test mode used to test static-Icc and pin leakage. The requirements for this mode are 1) all system logic outputs are high impedance, including TDO; 2) all pull-ups are disables, including TMS , TRST# , and TDI ; 3) all clocks are stopped. The TAP controller cannot be operational in this mode.



21.0 Peripheral Registers

This chapter summarizes the registers for the integrated peripherals. Each register is defined in detail in the corresponding unit chapter.

21.1 Overview

Intel® 81341 and 81342 I/O Processors (81341 and 81342) Peripheral Registers can be accessed via three methods:

- **PCI Configuration Register interface:** Is supported by the PCI interface and PCI Configuration Cycle transaction type.
- **Peripheral Memory-Mapped Register (PMMR) interface:** Gives software ability to read and modify internal control registers. These registers are accessed as a memory-mapped 32-bit register with unique memory address. Access is accomplished through regular Intel XScale® processor memory-format instruction.
- **High-performance Intel XScale® processor (ARM* architecture compliant) Coprocessor Register interface (CCR):** Gives software ability to read and modify internal control registers at very low latency as compared to PMMR interface.

These registers are specific to the 81341 and 81342 only. They support the:

• I/O Level Control	• Inter-Processor Messaging Unit
• Internal Bus Bridge Unit	• Address Translation Unit (PCI-X and PCI-E)
• DDR SDRAM Memory Controller	• Application DMA Controller Units
• I ² C Bus Interface Units	• Peripheral Performance Monitoring Unit
• Interrupt Controller Unit	• General Purpose I/O Unit
• Messaging Unit	• Peripheral Bus Interface Unit
• System Controllers	• Intel XScale® processor Bus Interface Unit
• UART Units	• Inter-Processor Communication
• SRAM Memory Controller	

Each of these peripherals fully describe the independent functionality of the registers, control and usage.

Control and status registers for the Intel XScale® processor use the CCR interface. Accesses to coprocessor registers do not generate external bus cycles. See the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual* (Order Number: 273411) for a full description of the usage of the coprocessor register space. For completeness, these registers are included in the tables of registers for the CCR interface. These registers can only be accessed using the coprocessor instructions.

The PMMR interface provides full accessibility from the ATU, and the Intel XScale® processor. The PMMR block can be mapped to any 512-KByte aligned address boundary using the PMMR Base Address registers. The default starting address of the PMMR block is 0 FFD8 0000H.



21.2 Accessing Peripheral Memory-Mapped Registers

The PMMR interface is a slave device connected to the 81341 and 81342 internal bus. This interface accepts data transactions which appear on the internal bus from the ATU and the Intel XScale® processors.

The PMMR interface allows these devices to perform read, write, or read-modify-write transactions. The specific actions taken when modifying any value in the PMMR space is independently defined within each chapter which describes the functionality of the register.

Note: The PMMR interface does not support multi-word burst accesses from any internal bus master.

All PMMR transactions are allowed from the Intel XScale® processors operating in user or supervisor mode. In addition, the PMMR does not provide any access exception to the Intel XScale® processor.

21.3 Accessing Peripheral Registers Using the Core Coprocessor Register Interface

Registers may be accessed/manipulated through the CCR interface with the MCR, MRC, STC, and LDC instructions. The *CRn* field of the instruction denotes the register number to be accessed. The *opcode_1* and *opcode_2* fields of the instruction should be zero. The *CRm* field must be set to 0 for the Interrupt Controller Unit and to 1 for the Programmable Timers. Most systems restrict access to coprocessor registers to privileged processes. To control access to a coprocessor register, use the Coprocessor Access Register as described in the *ARM Architecture Reference Manual*.

21.4 Architecturally Reserved Memory Space

The 81341 and 81342 provides 64 GBytes of address space. Portions of this address space is architecturally reserved and users are restricted as to their function. [Figure 174](#) shows the reserved address space.

Addresses 0 0000 0000H through 0 0000 001FH are reserved for the exception vectors of the Intel XScale® processor.

Addresses 0 FFFF 0000H through 0 FFFF 001FH are reserved for the relocated exception vectors of the Intel XScale® processor.²⁹

Addresses 0 FFD8 0000H through 0 FFDF FFFFH are allocated to the PMMR interface by default. These registers can be relocated by re-programming the PMMR Base Address Register (PMMRBAR).

29. By enabling the Exception Vector Relocation mode (bit 13, CP15, Register 1), the Exception Vectors (except Reset Vector at 0000 0000H) can be relocated to be based at FFFF 0000H rather than 0000 0000H. (i.e., FIQ Vector located at FFFF 001CH)



21.5 Default Memory Space Setup

Figure 174 shows the Intel XScale® processor address space and addresses available to the applications. Figure 174 also shows the default locations of certain resources after reset.

The PMMR Block occupies 512 KBytes of space and is located at addresses 0 FFD8 0000H through 0 FFDF FFFFH.

The Messaging Unit occupies 8 KBytes of space and is located at addresses 0 FF00 0000H through 0 FF00 1FFFH.

The ATU-X Outbound I/O Translation Window occupies 64 KBytes of space and is located at addresses 0 FFFB 0000H through 0 FFFB FFFFH.

The ATU-E Outbound I/O Translation Window occupies 64 KBytes of space and is located at addresses 0 FFFD 0000H through 0 FFFD FFFFH.

The SRAM Memory occupies 1 MByte of space and is located at addresses 0 FFE0 0000H through 0 FFEF FFFFH.

For a full description of the address space for the Intel XScale® processor Reset and Exception Vectors, refer to the *ARM Architecture Reference Manual*.



Figure 173. Memory Address Space

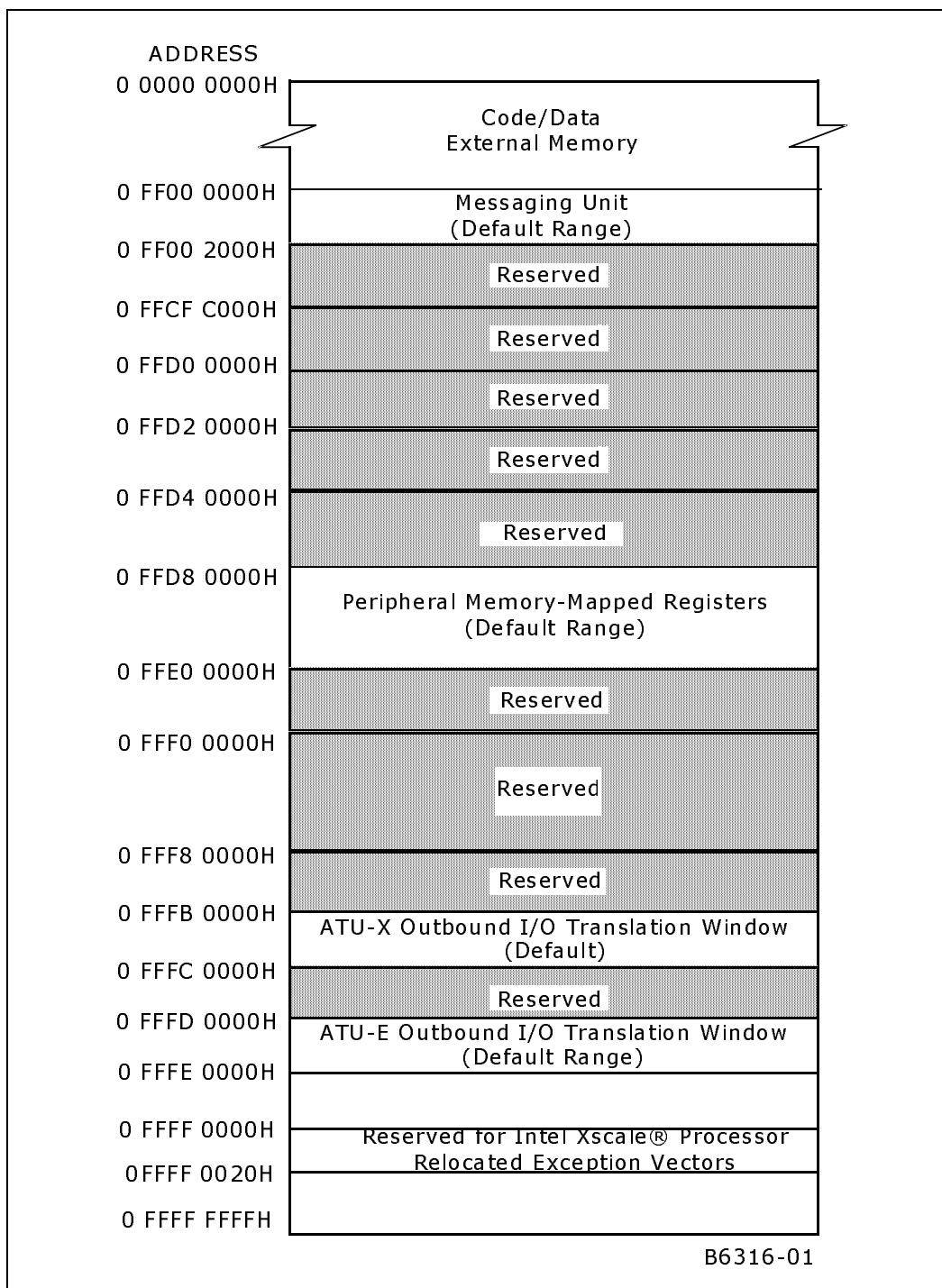
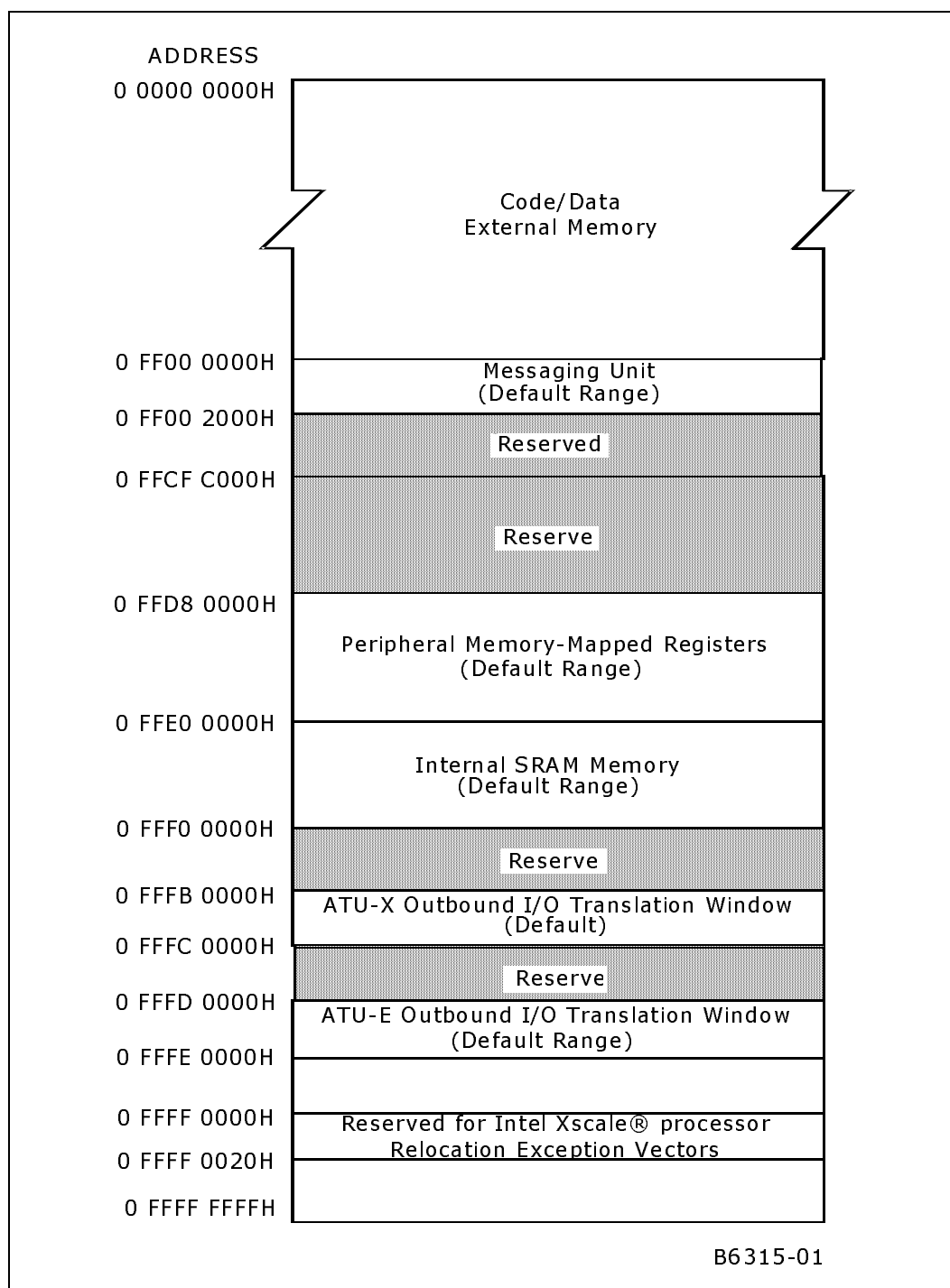




Figure 174. Intel® 81341 and 81342 I/O Processors Memory Address Space





21.6 Peripheral Memory-Mapped Register Address Space

The PMMR address space is divided to support the integrated peripherals on the 81341 and 81342. Table 638 shows all of the 81341 and 81342 integrated peripheral memory-mapped registers and their internal bus address offsets. The starting address of the memory-mapped registers is programmable using the PMMRBAR register. The PMMRBAR register is located at a fixed location which is at F_FFFF_FFFCH. The default starting address of the PMMR register block is at 0_FFD8_0000H. Software can reprogram the PMMRBAR register to relocate the memory-mapped register block when desired. Bits [31:15] of the PMMRBAR register represent bits [35:19] of the 36-bit base address. The PMMR register block is aligned on a 512 KByte boundary. Bits [14:0] of the PMMRBAR registers are reserved bits.

Table 638 lists the peripherals and the relative address offsets from the PMMRBAR content.

Warning: Care must be exercised when updating memory-mapped registers, because some of the registers can have side-effects on other registers and resources in the 81341 and 81342 architecture. As an example, updating a Base Address Register (such as the PMMRBAR) relocates other registers and resources in the 81341 and 81342 architecture, and software must not access these related registers and resources until software can ensure that the Base Address Register has truly been updated. Refer to the Intel XScale® Microarchitecture *Developer's Manual*, which describes various methods of fencing memory accesses.

Table 637. PMMR Base Address Register (PMMRBAR) Default Value

Register	Absolute Address
PMMRBAR	F_FFFF_FFFCH

Table 638. Local Addresses for Integrated Peripherals (Sheet 1 of 3)

Integrated Peripheral	Internal Address Offset (Relative to PMMRBAR)	Space Allocated
"Application DMA Unit" 0	+0000H through 01FFH	512 Bytes
"Application DMA Unit" 1	+0200H through 03FFH	512 Bytes
"Application DMA Unit" 2	+0400H through 05FFH	512 Bytes
Reserved.	+0600H through 07FFH	512 Bytes
Not Claimed by any Unit ^a	+0800H through 09FFH	x
"Inter-Processor Messaging Unit"	+0A00H through 0CFFH	768 Bytes
Not Claimed by any Unit	+0D00H through 14FFH	x
"SRAM Memory Controller"	+1500H through 157FH	128 Bytes
"Peripheral Bus Interface Unit"	+1580H through 15FFH	128 Bytes
Not Claimed by any Unit	+1600H through 163FH	x
"System Controller Unit"	+1640H through 164FH	16 Bytes
Not Claimed by any Unit	+1650H through 167FH	32 Bytes
Not Claimed by any Unit	+1680H through 16FFH	x
Not Claimed by any Unit	+1700H through 177FH	x
"Internal Bus Bridge"	+1780H through 1797H	24 Bytes
Not Claimed by any Unit	+1798H through 177FH	104 Bytes
"DDR SDRAM Memory Controller"	+1800H through 19FFH	512 Bytes
Not Claimed by any Unit	+1A00H through 1FFFFH	x
"I/O Pad Control Unit"	+2000H through 21FFH	512 Bytes
Reserved	+2200H through 221FH	32 Bytes

**Table 638. Local Addresses for Integrated Peripherals (Sheet 2 of 3)**

Integrated Peripheral	Internal Address Offset (Relative to PMMRBAR)	Space Allocated
Not Claimed by any Unit	+2220H through 22FFH	x
"UART" 0	+2300H through 233FH	64 Bytes
"UART" 1	+2340H through 237FH	64 Bytes
Not Claimed by any Unit	+2380H through 247FH	x
"GPIO"	+2480H through 24BFH	64 Bytes
Not Claimed by any Unit	+24C0H through 24FFH	x
"I2C Unit" 0	+2500H through 251FH	32 Bytes
"I2C Unit" 1	+2520H through 253FH	32 Bytes
"I2C Unit" 2	+2540H through 255FH	32 Bytes
Reserved.	+2560H through 257FH	32 Bytes
Not Claimed by any Unit	+2580H through 25FFH	x
Reserved	+2600H through 267FH	128 Bytes
Reserved	+2680H through 26FFH	128 Bytes
Not Claimed by any Unit	+2700H through 3FFFH	x
"Messaging Unit"	+4000H through 5FFFH	8 KBytes
Not Claimed	+6000H through 7FFFH	x
Reserved.	+1 8000H through 1 81FFH	512 Bytes
Reserved.	+1 8200H through 1 83FFH	512 Bytes
Not Claimed by any Unit	+1 8400H through 1 9FFFH	x
"PMON Unit"	+1 A000H through 1 BFFFH	8 KBytes
Not Claimed by any Unit	+1 C000H through 3 FFFFH	x
PCI Function 0 Configuration Registers (PCI Attributes)	+4 0000H through 4 0FFFH	4 KBytes
Reserved.	+4 1000H through 4 1FFFH	4 KBytes
Reserved.	+4 2000H through 4 2FFFH	4 KBytes
Reserved.	+4 3000H through 4 3FFFH	4 KBytes
Reserved.	+4 4000H through 4 4FFFH	4 KBytes
Reserved.	+4 5000H through 4 5FFFH	4 KBytes
Reserved	+4 6000H through 4 6FFFH	4 KBytes
Not Claimed by any Unit	+4 7000H through 4 7FFFH	4 KBytes
PCI Function 0 Configuration Registers (Local Attributes)	+4 8000H through 4 8FFFH	4 KBytes
Reserved.	+4 9000H through 4 9FFFH	4 KBytes
Reserved.	+4 A000H through 4 AFFFH	4 KBytes
Reserved.	+4 B000H through 4 BFFFH	4 KBytes
Reserved.	+4 C000H through 4 CFFFH	4 KBytes
Reserved.	+4 D000H through 4 DFFFH	4 KBytes
Reserved	+4 E000H through 4 EFFFH	4 KBytes
Not Claimed by any Unit	+4 F000H through 4 FFFFH	4 KBytes
Reserved	+5 0000H through 5 03FFH	1 KByte
Reserved	+5 0400H through 5 07FFH	1 KByte
Reserved	+5 0800H through 5 0BFFH	1 KByte
Reserved	+5 0C00H through 5 0FFFH	1 KByte
Not Claimed by any Unit	+5 1000H through 5 13FFH	1 KByte
Not Claimed by any Unit	+5 1400H through 5 17FFH	1 KByte
Not Claimed by any Unit	+5 1800H through 5 1BFFH	1 KByte
Not Claimed by any Unit	+5 1C00H through 5 1FFFH	1 KByte
Reserved	+5 2000H through 5 23FFH	1 KByte
Reserved	+5 2400H through 5 27FFH	1 KByte

**Table 638. Local Addresses for Integrated Peripherals (Sheet 3 of 3)**

Integrated Peripheral	Internal Address Offset (Relative to PMMRBAR)	Space Allocated
Reserved	+5 2800H through 5 2BFFH	1 KByte
Reserved	+5 2C00H through 5 2FFFH	1 KByte
Not Claimed by any Unit	+5 3000H through 5 33FFH	1 KByte
Not Claimed by any Unit	+5 3400H through 5 37FFH	1 KByte
Not Claimed by any Unit	+5 3800H through 5 3BFFH	1 KByte
Not Claimed by any Unit	+5 3C00H through 5 3FFFH	1 KByte
Reserved.	+6 0000H through 6 1FFFH	8 KBytes
Reserved.	+6 2000H through 6 3FFFH	8 KBytes
Reserved.	+6 4000H through 6 5FFFH	8 KBytes
Reserved.	+6 6000H through 6 7FFFH	8 KBytes
Not Claimed by any Unit	+6 8000H through 6 9FFFH	8 KBytes
Not Claimed by any Unit	+6 A000H through 6 BFFFH	8 KBytes
Not Claimed by any Unit	+6 C000H through 6 DFFFH	8 KBytes
Not Claimed by any Unit	+6 E000H through 6 FFFFH	8 KBytes
Reserved.	+7 0000H through 7 1FFFH	8 KBytes
Reserved.	+7 2000H through 7 3FFFH	8 KBytes
Reserved.	+7 4000H through 7 5FFFH	8 KBytes
Reserved.	+7 6000H through 7 7FFFH	8 KBytes
Not Claimed by any Unit	+7 8000H through 7 9FFFH	8 KBytes
Not Claimed by any Unit	+7 A000H through 7 BFFFH	8 KBytes
Not Claimed by any Unit	+7 C000H through 7 DFFFH	8 KBytes
Not Claimed by any Unit	+7 E000H through 7 FFFFH	8 KBytes
Not Claimed by any Unit	+8 0000H through 8 FFFFH	x

a. Address Ranges that are defined as “Not Claimed by any Unit” Master Aborts when accessed.

Memory-mapped registers also accessible via PCI configuration transactions are:

- Address Translation Unit (PCI-X)
- Address Translation Unit (PCI-E)
- Messaging Unit (MU)

Registers which must have address translation logic configured to translate PCI addresses into the Intel XScale® processor address space, to access the memory-mapped registers from the PCI Express interface are:

- Application DMA Controllers
- Address Translation Unit (PCI-E)
- DDR SDRAM Memory Controller
- SRAM Memory Controller
- I²C Bus Interface Unit
- Messaging Unit
- System Controllers
- Peripheral Bus Interface Unit
- Performance Monitoring Unit
- Interrupt Controller Unit
- General Purpose I/O Unit
- Interface Pad Control Registers
- Third Party Messaging Unit
- Inter-Processor Messaging Unit
- UARTs

The following sections describe the register map for the internal units of the Intel® 81341 and 81342. See the relevant unit chapters for detailed description of register function.



21.6.1 Internal Units

21.6.1.1 Application DMA 0-2

The 81341 and 81342 contains three instances of the Application DMA (ADMA). These three units have individual 512 Byte PMMR Register space that is accessible on the internal bus. The registers offsets shown in [Table 640](#) are relative to the ADMA base address offset shown below in [Table 639](#)

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + ADMAx Base Address Offset + Register Offset.

Table 639. ADMAx Base Address Offset.

Unit	ADMAx Base Address Offset (Relative to PMMRBAR)
ADMA0	+000H
ADMA1	+200H
ADMA2	+400H

Table 640. Application DMA Unit (Sheet 1 of 2)

Register Description (Name)	Register Size in Bits	ADMA Register Offset (Relative to ADMAx Base Address Offset)
ADMA Channel Control Register x — ACCRx	32	+000
ADMA Channel Status Register x — ACSRx	32	+004
ADMA Descriptor Address Register x — ADARx	32	+008
Reserved	x	+00C through 014
Internal Interface Parity Control Register x — IIPCRx	32	+018
Reserved	32	+01C
Reserved	32	+020
ADMA Next Descriptor Address Register x — ANDARx	32	+024
ADMA Descriptor Control Register x — ADCRx	32	+028
CRC Address/Memory Block Fill Data Register x — CARMDx	32	+02C
ADMA Byte Count Register x — ABCRx	32	+030
Destination Lower Address Register x — DLADRx	32	+034
Destination Upper Address Register x — DUADRx	32	+038
Source Lower Address Register 0_x — SLAR0_x	32	+03C
Source Upper Address Register 0_x — SUAR0_x	32	+040
Source Lower Address Register 1_x — SLAR1_x	32	+044
Source Upper Address Register 1_x — SUAR1_x	32	+048
Source Lower Address Register 2_x — SLAR2_x	32	+04C
Source Upper Address Register 2_x — SUAR2_x	32	+050
Source Lower Address Register 3_x — SLAR3_x	32	+054
Source Upper Address Register 3_x — SUAR3_x	32	+058
Source Lower Address Register 4_x — SLAR4_x	32	+05C
Source Upper Address Register 4_x — SUAR4_x	32	+060

**Table 640. Application DMA Unit (Sheet 2 of 2)**

Register Description (Name)	Register Size in Bits	ADMA Register Offset (Relative to ADMAx Base Address Offset)
Source Lower Address Register 5_x — SLAR5_x	32	+064
Source Upper Address Register 5_x — SUAR5_x	32	+068
Source Lower Address Register 6_x — SLAR6_x	32	+06C
Source Upper Address Register 6_x — SUAR6_x	32	+070
Source Lower Address Register 7_x — SLAR7_x	32	+074
Source Upper Address Register 7_x — SUAR7_x	32	+078
Source Lower Address Register 8_x — SLAR8_x	32	+07C
Source Upper Address Register 8_x — SUAR8_x	32	+080
Source Lower Address Register 9_x — SLAR9_x	32	+084
Source Upper Address Register 9_x — SUAR9_x	32	+088
Source Lower Address Register 10_x — SLAR10_x	32	+08C
Source Upper Address Register 10_x — SUAR10_x	32	+090
Source Lower Address Register 11_x — SLAR11_x	32	+094
Source Upper Address Register 11_x — SUAR11_x	32	+098
Source Lower Address Register 12_x — SLAR12_x	32	+09C
Source Upper Address Register 12_x — SUAR12_x	32	+0A0
Source Lower Address Register 13_x — SLAR13_x	32	+0A4
Source Upper Address Register 13_x — SUAR13_x	32	+0A8
Source Lower Address Register 14_x — SLAR14_x	32	+0AC
Source Upper Address Register 14_x — SUAR14_x	32	+0B0
Source Lower Address Register 15_x — SLAR15_x	32	+0B4
Source Upper Address Register 15_x — SUAR15_x	32	+0B8
Reserved	x	+0BC through 1FF



21.6.1.2 Inter-Processor Messaging Unit

The Inter-Processor Messaging Unit is allocated 768 Bytes of PMMR Register space. It is always located at offset +A00H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + IMU Base Address Offset + Register Offset.

Table 641. IMU Base Address Offset.

Unit	Associated Interface	IMU Base Address Offset (Relative to PMMRBAR)
IMU	IMU Control Registers	+A00H
	IMU Test and Set Registers	+B00H

Table 642. Inter-Processor Messaging Unit (Sheet 1 of 2)

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to IMU Base Address)
Door Bell Control Register — DBCR	32	+00H
Door Bell Enable Register — DBER	32	+04H
Reserved	32	+08H
Reserved	32	+0CH
Door Bell Assertion Register — DBAR	32	+10H
Door Bell Enable Other Processor Register — DBEOR	32	+14H
Reserved	32	+18H
Reserved	32	+1CH
Send Queue Put/Get Pointer Register 0 — SQPG0	32	+20H
Send Queue Control Register 0 — SQCR0	32	+24H
Send Queue Lower Base Address Register 0 — SQLBAR0	32	+28H
Send Queue Upper Base Address Register 0 — SQUBAR0	32	+2CH
Receive Queue Put/Get Pointer Register 0 — RQPG0	32	+30H
Receive Queue Control Register 0 — RQCR0	32	+34H
Receive Queue Lower Base Address Register 0 — RQLBAR0	32	+38H
Receive Queue Upper Base Address Register 0 — RQUBAR0	32	+3CH
Send Queue Put/Get Pointer Register 1 — SQPG1	32	+40H
Send Queue Control Register 1 — SQCR1	32	+44H
Send Queue Lower Base Address Register 1 — SQLBAR1	32	+48H
Send Queue Upper Base Address Register 1 — SQUBAR1	32	+4CH
Receive Queue Put/Get Pointer Register 1 — RQPG1	32	+50H
Receive Queue Control Register 1 — RQCR1	32	+54H
Receive Queue Lower Base Address Register 1 — RQLBAR1	32	+58H
Receive Queue Upper Base Address Register 1 — RQUBAR1	32	+5CH
Send Queue Put/Get Pointer Register 2 — SQPG2	32	+60H
Send Queue Control Register 2 — SQCR2	32	+64H
Send Queue Lower Base Address Register 2 — SQLBAR2	32	+68H

**Table 642. Inter-Processor Messaging Unit (Sheet 2 of 2)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to IMU Base Address)
Send Queue Upper Base Address Register 2 — SQUBAR2	32	+6CH
Receive Queue Put/Get Pointer Register 2 — RQPG2	32	+70H
Receive Queue Control Register 2 — RQCR2	32	+74H
Receive Queue Lower Base Address Register 2 — RQLBAR2	32	+78H
Receive Queue Upper Base Address Register 2 — RQUBAR2	32	+7CH
Send Queue Put/Get Pointer Register 3 — SQPG3	32	+80H
Send Queue Control Register 3 — SQCR3	32	+84H
Send Queue Lower Base Address Register 3 — SQLBAR3	32	+88H
Send Queue Upper Base Address Register 3 — SQUBAR3	32	+8CH
Receive Queue Put/Get Pointer Register 3 — RQPG3	32	+90H
Receive Queue Control Register 3 — RQCR3	32	+94H
Receive Queue Lower Base Address Register 3 — RQLBAR3	32	+98H
Receive Queue Upper Base Address Register 3 — RQUBAR3	32	+9CH
Reserved.	x	+A0H through AFH
IMU Test and Set Registers — IMUTSR[0:511]	8 x 512	+000H through 1FFH



21.6.1.3 SRAM Memory Controller

The SRAM Memory Controller is allocated 128 Bytes of PMMR registers space and is always located at offset +1500H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + SMCU Base Address Offset + Register Offset.

Table 643. SMCU Base Address Offset.

Unit	SMCU Base Address Offset (Relative to PMMRBAR)
SRAMMCU	+1500H

Table 644. SRAM Memory Controller

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to SMCU Base Address Offset)
SRAM Base Address Register — SRAMBAR	32	+00H
SRAM Upper Base Address Register — SRAMUBAR	32	+04H
SRAM ECC Control Register — SECR	32	+08H
SRAM ECC Log Register — SELOGR	32	+0CH
SRAM ECC Address Register — SEAR	32	+10H
SRAM ECC Context Register — SECAR	32	+14H
SRAM ECC Test Register — SECTST	32	+18H
SRAM Parity Control and Status Register — SPARCSR	32	+1CH
SRAM Parity Address Register — SPAR	32	+20H
SRAM Parity Upper Address Register — SPUAR	32	+24H
SRAM Parity Context Address Register — SPCAR	32	+28H
SRAM Memory Controller Interrupt Status Register — SMCISR	32	+2CH
Reserved	32	+30H
Reserved	32	+34
Reserved	x	+38H through +7FH



21.6.1.4 Peripheral Bus Interface Unit

The Peripheral Bus Interface Unit (PBI) is allocated 128 Bytes of PMMR registers space and is always located at offset +1580H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

$$\text{Internal Bus Address} = \text{PMMRBAR} + \text{PBI Base Address Offset} + \text{Register Offset.}$$

Note: Additionally, GPIO[8:0] I/O pad control registers are located in the “I/O Pad Control Unit” registers block.

Table 645. PBI Base Address Offset.

Unit	PBI Base Address Offset (Relative to PMMRBAR)
PBI	+1580H

Table 646. Peripheral Bus Interface Unit

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to PBI Base Address Offset)
PBI Control Register — PBCR	32	+00H
PBI Status Register — PBISR	32	+04H
PBI Base Address Register 0 — PBBAR0	32	+08H
PBI Limit Register 0 — PBLR0	32	+0CH
PBI Base Address Register 1 — PBBAR1	32	+10H
PBI Limit Register 1 — PBLR1	32	+14H
Reserved	x	+18H through +7FH

21.6.1.5 System Controller

The System Controller Unit (SC) is allocated 16 Bytes of PMMR register space and is always located at offset +1640H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

$$\text{Internal Bus Address} = \text{PMMRBAR} + \text{SC Base Address Offset} + \text{Register Offset.}$$

Table 647. SC Base Address Offset.

Unit	SC Base Address Offset (Relative to PMMRBAR)
System Controller	+1640H

Table 648. System Controller Unit

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to SC Base Address Offset)
Internal Bus Arbitration Control Register — IBACR	32	+00H
South Internal Bus Address Test Register — SIBATCR	32	+04H
South Internal Bus Data Test Register — SIBDTCR	32	+08H
Reserved	x	+0CH through 0FH
Peripheral Memory-Mapped Register Base Address Register (PMMRBAR)	32	F FFF FFFCH (absolute address)



21.6.1.6 Internal Bus Bridge

The Internal Bus Bridge is allocated 24 Bytes of PMMR register space and is always located at offset +1780H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + Internal Bus Bridge Address Offset + Register Offset.

Table 649. Internal Bus Bridge Base Address Offset.

Unit	Internal Bus Bridge Base Address Offset (Relative to PMMRBAR)
Internal Bus Bridge	+1780H

Table 650. Internal Bus Bridge

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to Internal Bus Bridge Base Address Offset)
Bridge Window Base Address Register — BWBAR	32	+00H
Bridge Window Upper Base Address Register — BWUBAR	32	+04H
Bridge Window Limit Register — BWLR	32	+08H
Bridge Error Status Register — BECSR	32	+0CH
Bridge Error Address Register — BERAR	32	+10H
Bridge Error Upper Address Register — BERUAR	32	+14H

**21.6.1.7 DDR SDRAM Memory Controller**

DDR SDRAM Memory Controller is allocated 512 Bytes of PMMR registers space and is always located at offset +1800H relative to the PMMRBAR. Use the following equation to calculate the actual register address:

$$\text{Internal Bus Address} = \text{PMMRBAR} + \text{DMCU Base Address Offset} + \text{Register Offset.}$$

Note: DDR I/O pad control registers are located in “I/O Pad Control Unit” registers block.

Table 651. DMCU Base Address Offset.

Unit	DMCU Base Address Offset (Relative to PMMRBAR)
DMCU	+1800H

Table 652. DDR SDRAM Memory Controller

Register Description (Name)	Bits	Internal Bus Address Offset ^a
SDRAM Initialization Register — SDIR	32	+000H
SDRAM Control Register 0 — SDCR0	32	+004H
SDRAM Control Register 1 — SDCR1	32	+008H
SDRAM Base Register — SDBR	32	+00CH
SDRAM Upper Base Register — SDUBR	32	+010H
SDRAM Bank Size Register — SBSR	32	+014H
SDRAM 32-bit Region Size Register — S32SR	32	+018H
DDR ECC Control Register — DECCR	32	+01CH
DDR ECC Log 0 Register — DELOG0	32	+020H
DDR ECC Log 1 Register — DELOG1	32	+024H
DDR ECC Address 0 Register — DEAR0	32	+028H
DDR ECC Address 1 Register — DEAR1	32	+02CH
DDR ECC Context Address 0 Register — DECAR0	32	+030H
DDR ECC Context Address 1 Register — DECAR1	32	+034H
DDR ECC Context Upper Address 0 Register — DECUAR0	32	+038H
DDR ECC Context Upper Address 1 Register — DECUAR1	32	+03CH
ECC Test Register — DECTST	32	+040H
DDR Parity Control and Status Register — DPCSR	32	+044H
DDR Parity Address Register — DPAR	32	+048H
DDR Parity Upper Address Register — DPUAR	32	+04CH
DDR Parity Context Address Register — DPCAR	32	+050H
Reserved	32	+054H
DDR Parity Context Upper Address Register — DPCUAR	32	+058H
DDR Memory Controller Interrupt Status Register — DMCISR	32	+060H
Reserved	32	+064H
DDR MCU Port Transaction Count Register — DMPTCR	32	+068H
DDR MCU Preemption Control Register — DMPCR	32	+06CH
Refresh Frequency Register — RFR	32	+070H
Reserved	x	+074H through +0B0H
Reserved	x	+0B4H through +1FFH

a. Relative to DMCU base address offset.



21.6.1.8 I/O Pad Control

The I/O Pad Control is allocated 512 Bytes of PMMR registers space and is always located at offset +2000H relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + I/O Pad Control Base Address Offset + Register Offset.

Note: DCAL and DQS registers are described in the DDR SDRAM Memory Controller Unit, and the PBI drive strength register is described in the Peripheral Bus Interface chapter.

Table 653. I/O Pad Control Base Address Offset.

Unit	Associated Unit Interface	I/O Pad Control Base Address Offset (Relative to PMMRBAR)
I/O Pad Control	DDR SDRAM Memory Controller	+2000H
	Peripheral Bus Interface	+2080H
	PCI Interface	+2100H
	Other Units	+2180H



Table 654. I/O Pad Control Unit (Sheet 1 of 2)

Unit	Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to I/O Pad Control Base Address Offset)
DDR SDRAM Memory Controller	DDR RCOMP Control Register — DRCCR	32	+00H
	RCOMP Pad Drive Strength Select Mux — RPDSR	32	+04H
	DQ Pad ODT Drive Strength Manual Override Values Register — DQPODSR	32	+08H
	DQ Pad Drive Strength Manual Override Values Register — DQPDSR	32	+0CH
	AD Pad Drive Strength Manual Override Values Register — ADPDSR	32	+10H
	MCLK Pad Drive Strength Manual Override Values Register — MPDSR	32	+14H
	CKE/CS Pad Drive Strength Manual Override Values Register — CKEPDSR	32	+18H
	DLL Delay for DQS0, DQS0#, DQS1 and DQS1# Clock Register 0 — DLLR0	32	+1CH
	DLL Delay for DQS2, DQS2#, DQS3 and DQS3# Clock Register 1 — DLLR1	32	+20H
	DLL Delay for DQS4, DQS4#, DQS5 and DQS5# Clock Register 2 — DLLR2	32	+24H
	DLL Delay for DQS6, DQS6#, DQS7 and DQS7# Clock Register 3 — DLLR3	32	+28H
	DLL Delay DQS8, DQS8# clock, Master DLL Delay Block for DQS Strokes and Receive Enable Pad Master DLL block Register 4 — DLLR4	32	+2CH
	DLL Delay for Receive Enable Register — DLLRCVER	32	+30H
	Reserved	x	+34H through +7FH
Peripheral Bus Interface	PBI Drive Strength Control Register — PBDSCR	32	+00H
	Reserved	x	+04H through +7FH
PCI Interface	PCIX RCOMP Control Register — PRCR	32	+00H
	PCIX Pad ODT Drive Strength Manual Override Values Registers — PPODSMOVR	32	+04H
	PCIX PAD DRIVE STRENGTH manual override values register (3.3V/1.5V switch supply voltage) — PPDSMOVR3.3_1.5	32	+08H
	PCIX PAD DRIVE STRENGTH manual override values register(3.3V dedicated supply voltage) — PPDSMOVR3.3	32	+0CH
	Reserved	x	+10H through +7FH

Notes:

1. Registers that belong in this group are documented in the Peripheral Bus Interface Unit chapter.

**Table 654. I/O Pad Control Unit (Sheet 2 of 2)**

Unit	Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to I/O Pad Control Base Address Offset)
Other Units ¹	Reserved	32	+00H
	Reserved	32	+04H
	Reserved	32	+08H
	Reserved	32	+0CH
	Reserved	32	+10H
	Unique ID Register 0 — UID0	32	+14H
	Unique ID Register 1 — UID1	32	+18H
	Reserved	x	+1CH through +7FH

Notes:

1. Registers that belong in this group are documented in the Peripheral Bus Interface Unit chapter.



21.6.1.9 UART 0-1

The 81341 and 81342 contains two instances of the UART. Each UART is allocated 64 Bytes of PMMR registers space that is located at the offset specified in [Table 655](#) which is relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + UART Base Address Offset + Register Offset.

Table 655. UART 0-1 Offset.

Unit	UARTx Base Address Offset (Relative to PMMRBAR)
UART 0	+2300H
UART 1	+2340H

Table 656. UART

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to UARTx Base Address Offset)
UART x Receive Buffer Register (Read Only) (DLAB=0)	32	+00H
UART x Transmit Holding Register (Write Only) (DLAB=0)	32	
UART x baud Divisor Latch Low byte (DLAB=1)	8	
UART x Interrupt Enable Register (DLAB=0)	8	+04H
UART x baud Divisor Latch High byte (DLAB=1)	8	
UART x Interrupt ID Register (Read Only)	8	+08H
UART x FIFO Control Register (Write Only)	8	
UART x Line Control Register	8	+0CH
UART x Modem Control Register	8	+10H
UART x Line Status Register	8	+14H
UART x Modem Status Register	8	+18H
UART x Scratch Pad Register	8	+1CH
Reserved	32	+20H
UART x FIFO Occupancy Register	8	+24H
UART x Autobaud Control Register	8	+28H
UART x Autobaud Count Register	16	+2CH
Reserved		+30H through +3FH



21.6.1.10 GPIO

The GPIO block is allocated 64Bytes of PMMR registers space that is located at the offset specified in [Table 658](#) which is relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

$$\text{Internal Bus Address} = \text{PMMRBAR} + \text{GPIO Base Address Offset} + \text{Register Offset.}$$

Table 657. GPIO Offset.

Unit	GPIO Base Address Offset Relative to PMMRBAR)
GPIO	+2480H

Table 658. GPIO

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to GPIO Base Address Offset)
GPIO Output Enable Register — GPOE	32	+00H
GPIO Input Data Register — GPID	32	+04H
GPIO Output Data Register — GPOD	32	+08H
Reserved	x	+0CH–3FH

21.6.1.11 I²C Bus Interface Unit 0-2

The 81341 and 81342 contains three instances of the I²C Unit which are each allocated 32 Bytes of PMMR registers space located at the offset specified in [Table 660](#).

Use the following equation to calculate the actual register address:

$$\text{Internal Bus Address} = \text{PMMRBAR} + \text{I}^2\text{C Base Address Offset} + \text{Register Offset.}$$

Table 659. I²C 0-2 Offset.

Unit	I ² C Base Address Offset (Relative to PMMRBAR)
I ² C 0	+2500H
I ² C 1	+2520H
I ² C 2	+2540H

Table 660. I²C Unit

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to I ² C Base Address Offset)
I ² C Control Register x — ICRx	32	+00H
I ² C Status Register x — ISRx	32	+04H
I ² C Slave Address Register x — ISARx	32	+08H
I ² C Data Buffer Register x — IBDRx	32	+0CH
Reserved	32	+10H
I ² C Bus Monitor Register x — IBMRx	32	+14H
I ² C Manual Bus Control Register x — IMBCRx	32	+18H
Reserved	32	+1CH



21.6.1.12 Messaging Unit

The Messaging Unit (MU) is allocated 8 KBytes of PMMR registers space that is located at the offset specified in [Table 661](#) which is relative to the PMMRBAR.

Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + MU Base Address Offset + Register Offset.

Table 661. Messaging Unit Offset.

Unit	MU Base Address Offset (Relative to PMMRBAR)
Messaging Unit	+4000H

Table 662. Messaging Unit (Sheet 1 of 2)

Register Description (Name)	Bits	Internal Bus Address Offset ^a
Reserved	x	+000H — +00C
Inbound Message Register — IMR0	32	+010H
Inbound Message Register — IMR1	32	+014H
Outbound Message Register — OMR0	32	+018H
Outbound Message Register — OMR1	32	+01CH
Inbound Doorbell Register — IDR	32	+020H
Inbound Interrupt Status Register — IISR	32	+024H
Inbound Interrupt Mask Register — IIMR	32	+028H
Outbound Doorbell Register — ODR	32	+02CH
Outbound Interrupt Status Register — OISR	32	+030H
Outbound Interrupt Mask Register — OIMR	32	+034H
Inbound Reset Control and Status Register — IRCSR	32	+038H
Outbound Reset Control and Status Register — ORCSR	32	+03CH
Reserved	x	+040H — +047H
MSI Inbound Message Register	32	+048H
Reserved	x	+04CH
MU Configuration Register — MUCR	32	+050H
Queue Base Address Register — QBAR	32	+054H
Reserved	32	+058H
Reserved	32	+05CH
Inbound Free Head Pointer Register — IFHPR	32	+060H
Inbound Free Tail Pointer Register — IFTPR	32	+064H
Inbound Post Head Pointer Register — IPHPR	32	+068H
Inbound Post Tail Pointer Register — IPTPR	32	+06CH
Outbound Free Head Pointer Register — OFHPR	32	+070H
Outbound Free Tail Pointer Register — OFTPR	32	+074H
Outbound Post Head Pointer Register — OPHPR	32	+078H
Outbound Post Tail Pointer Register — OPTPR	32	+07CH
Index Address Register — IAR	32	+080H
MU Base Address Register — MUBAR	32	+084H
MU Upper Base Address Register — MUUBAR	32	+088H
Reserved	x	+08CH — +0FFH

**Table 662. Messaging Unit (Sheet 2 of 2)**

Register Description (Name)	Bits	Internal Bus Address Offset ^a
MU MSI-X Table Message Address Register 0 — M_MT_MAR0	32	+1000
MU MSI-X Table Message Upper Address Register 0 — M_MT_MUAR0	32	+1004
MU MSI-X Table Message Data Register 0 — M_MT_MDR0	32	+1008
MU MSI-X Table Message Vector Control Register 0 — M_MT_MVCR0	32	+100C
MU MSI-X Table Message Address Register 1 — M_MT_MAR1	32	+1010
MU MSI-X Table Message Upper Address Register 1 — M_MT_MUAR1	32	+1014
MU MSI-X Table Message Data Register 1 — M_MT_MDR1	32	+1018
MU MSI-X Table Message Vector Control Register 1 — M_MT_MVCR1	32	+101C
MU MSI-X Table Message Address Register 2 — M_MT_MAR2	32	+1020
MU MSI-X Table Message Upper Address Register 2 — M_MT_MUAR2	32	+1024
MU MSI-X Table Message Data Register 2 — M_MT_MDR2	32	+1028
MU MSI-X Table Message Vector Control Register 2 — M_MT_MVCR2	32	+102C
MU MSI-X Table Message Address Register 3 — M_MT_MAR3	32	+1030
MU MSI-X Table Message Upper Address Register 3 — M_MT_MUAR3	32	+1034
MU MSI-X Table Message Data Register 3 — M_MT_MDR3	32	+1038
MU MSI-X Table Message Vector Control Register 3 — M_MT_MVCR3	32	+103C
MU MSI-X Table Message Address Register 4 — M_MT_MAR4	32	+1040
MU MSI-X Table Message Upper Address Register 4 — M_MT_MUAR4	32	+1044
MU MSI-X Table Message Data Register 4 — M_MT_MDR4	32	+1048
MU MSI-X Table Message Vector Control Register 4 — M_MT_MVCR4	32	+104C
MU MSI-X Table Message Address Register 5 — M_MT_MAR5	32	+1050
MU MSI-X Table Message Upper Address Register 5 — M_MT_MUAR5	32	+1054
MU MSI-X Table Message Data Register 5 — M_MT_MDR5	32	+1058
MU MSI-X Table Message Vector Control Register 5 — M_MT_MVCR5	32	+105C
MU MSI-X Table Message Address Register 6 — M_MT_MAR6	32	+1060
MU MSI-X Table Message Upper Address Register 6 — M_MT_MUAR6	32	+1064
MU MSI-X Table Message Data Register 6 — M_MT_MDR6	32	+1068
MU MSI-X Table Message Vector Control Register 6 — M_MT_MVCR6	32	+106C
MU MSI-X Table Message Address Register 7 — M_MT_MAR7	32	+1070
MU MSI-X Table Message Upper Address Register 7 — M_MT_MUAR7	32	+1074
MU MSI-X Table Message Data Register 7 — M_MT_MDR7	32	+1078
MU MSI-X Table Message Vector Control Register 7 — M_MT_MVCR7	32	+107C
Reserved	x	+1080 — +17FF
MU MSI-X Pending Bits Array Register	32	+1800
Reserved	x	+1804 — +1FFF

a. Relative to MU Base Address Offset.



21.6.1.13 PMON Unit

The **PMON** Unit (**PMON**) is allocated 8 KBytes of PMMR registers space that is located at the offset specified in [Table 663](#) which is relative to the PMMRBAR. Use the following equation to calculate the actual register address:

Internal Bus Address = PMMRBAR + **PMON** Base Address Offset + Register Offset.

Table 663. PMON Unit Base Address Offset.

Unit	PMON Base Address Offset (Relative to PMMRBAR)
PMON Unit	+1_A000H

Table 664. PMON Unit

Register Description (Name)	Register Size (Bits)	Internal Bus Address Offset (Relative to PMON Base Address Offset)
PMON Command Register 0 — PMON_CMD0	32	+00H
PMON Event Register 0 — PMON_EVR0	32	+04H
PMON Status Register 0 — PMON_STS0	32	+08H
PMON DATA Register 0 — PMON_DATA0	32	+0CH
PMON Command Register 1 — PMON_CMD1	32	+10H
PMON Event Register 1 — PMON_EVR1	32	+14H
PMON Status Register 1 — PMON_STS1	32	+18H
PMON DATA Register 1 — PMON_DATA1	32	+1CH
PMON Command Register 2 — PMON_CMD2	32	+20H
PMON Event Register 2 — PMON_EVR2	32	+24H
PMON Status Register 2 — PMON_STS2	32	+28H
PMON DATA Register 2 — PMON_DATA2	32	+2CH
PMON Command Register 3 — PMON_CMD3	32	+30H
PMON Event Register 3 — PMON_EVR3	32	+34H
PMON Status Register 3 — PMON_STS3	32	+38H
PMON DATA Register 3 — PMON_DATA3	32	+3CH
PMON Command Register 4 — PMON_CMD4	32	+40H
PMON Event Register 4 — PMON_EVR4	32	+44H
PMON Status Register 4 — PMON_STS4	32	+48H
PMON DATA Register 4 — PMON_DATA4	32	+4CH
PMON Command Register 5 — PMON_CMD5	32	+50H
PMON Event Register 5 — PMON_EVR5	32	+54H
PMON Status Register 5 — PMON_STS5	32	+58H
PMON DATA Register 5 — PMON_DATA5	32	+5CH
PMON Command Register 6 — PMON_CMD6	32	+60H
PMON Event Register 6 — PMON_EVR6	32	+64H
PMON Status Register 6 — PMON_STS6	32	+68H
PMON DATA Register 6 — PMON_DATA6	32	+6CH
PMON Command Register 7 — PMON_CMD7	32	+70H
PMON Event Register 7 — PMON_EVR7	32	+74H
PMON Status Register 7 — PMON_STS7	32	+78H
PMON DATA Register 7 — PMON_DATA7	32	+7CH



21.6.2 Host Interface Units

This section describes the register layout of the units that are visible as PCI functions. These units include the ATUE and ATUX. The PCI Function number for each of these units vary based on strapping options that are sampled during reset.

The PCI Function number associated with each unit and its Base Address Offset are detailed in Table 665, “PCI Function MMR Locations”

Table 665. PCI Function MMR Locations

PCI Function Number	INTERFACE_SEL_PCIX# =		Internal Bus Address Offset (Relative to PMMRBAR)	
	0	1	PCI Attributes	Local Attributes
0	ATUX	ATUE	+4 0000H	+4 8000H
1	Reserved	Reserved	+4 1000H	+4 9000H
2	Reserved	Reserved	+4 2000H	+4 A000H
3	Reserved	Reserved	+4 3000H	+4 B000H
4	Reserved	Reserved	+4 4000H	+4 C000H
5	Reserved	Reserved	+4 5000H	+4 D000H
6	PMON Reserved	PMON Reserved	+4 6000H	+4 E000H
7	Reserved	Reserved	+4 7000H	+4 F000H



21.6.2.1 Address Translation Unit (PCI-X)

A subset of the ATU registers are accessible through both inbound PCI configuration cycles and the 81341 and 81342 core CPU (Register offsets 000H through 0FFH). The balance of the registers are accessible only via the internal bus.

The Internal Bus Address Offset to PMMRBAR of any ATU Register can be derived by adding the 4 KB address aligned Internal Bus Memory Mapped Register Range Offset (Table 666, “81341 and 81342 ATUX Configuration Space Base Address Offset” on page 1042) to the Register Offset (Table 667, “Address Translation Unit Registers — ATUX” on page 1043)

For example, when INTERFACE_SEL_PCIX# is asserted, the offset to PMMRBAR of the ATU Command Register would be (4 C000H+004H) or 4 C004H.

Note: The 4 KB Address Aligned Range Offset can be different depending on two configuration straps as described in Table 666.

Table 666. 81341 and 81342 ATUX Configuration Space Base Address Offset

INTERFACE_SEL_PCIX#	ATUX Base Address Offset (Relative to PMMRBAR)
PCI Attributes	
Asserted	+4 0000H
Deasserted	+4 4000H
Local Attributes	
Asserted	+4 8000H
Deasserted	+4 C000H

**Table 667. Address Translation Unit Registers — ATUX (Sheet 1 of 3)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUX Base Address Offset)
ATU Vendor ID Register — ATUVID	16	+000H
ATU Device ID Register — ATUDID	16	+002H
ATU Command Register — ATUCMD	16	+004H
ATU Status Register — ATUSR	16	+006H
ATU Revision ID Register — ATURID	8	+008H
ATU Class Code Register — ATUCCR	24	+009H
ATU Cacheline Size Register — ATUCLSR	8	+00CH
ATU Latency Timer Register — ATULT	8	+00DH
ATU Header Type Register — ATUHTR	8	+00EH
ATU BIST Register — ATUBISTR	8	+00FH
Inbound ATU Base Address Register 0 — IABAR0	32	+010H
Inbound ATU Upper Base Address Register 0 — IAUBAR0	32	+014H
Inbound ATU Base Address Register 1 — IABAR1	32	+018H
Inbound ATU Upper Base Address Register 1 — IAUBAR1	32	+01CH
Inbound ATU Base Address Register 2 — IABAR2	32	+020H
Inbound ATU Upper Base Address Register 2 — IAUBAR2	32	+024H
Reserved	32	+028H
ATU Subsystem Vendor ID Register — ASVIR	16	+02CH
ATU Subsystem ID Register — ASIR	16	+02EH
Expansion ROM Base Address Register — ERBAR	32	+030H
ATU Capabilities Pointer Register — ATU_Cap_Ptr	8	+034H
Reserved	24	+035H
Reserved	32	+038H
ATU Interrupt Line Register — ATUILR	8	+03CH
ATU Interrupt Pin Register — ATUIPR	8	+03DH
ATU Minimum Grant Register — ATUMGNT	8	+03EH
ATU Maximum Latency Register — ATUMLAT	8	+03FH
Inbound ATU Limit Register 0 — IALR0	32	+040H
Inbound ATU Translate Value Register 0 — IATVR0	32	+044H
Inbound ATU Upper Translate Value Register 0 — IAUTVR0	32	+048H
Inbound ATU Limit Register 1 — IALR1	32	+04CH
Inbound ATU Translate Value Register 1 — IATVR1	32	+050H
Inbound ATU Upper Translate Value Register 1 — IAUTVR1	32	+054H
Inbound ATU Limit Register 2 — IALR2	32	+058H
Inbound ATU Translate Value Register 2 — IATVR2	32	+05CH
Inbound ATU Upper Translate Value Register 2 — IAUTVR2	32	+060H
Expansion ROM Limit Register — ERLR	32	+064H
Expansion ROM Translate Value Register — ERTVR	32	+068H
Expansion ROM Upper Translate Value Register — ERUTVR	32	+06CH
ATU Configuration Register — ATUCR	32	+070H
PCI Configuration and Status Register — PCSR	32	+074H
ATU Interrupt Status Register — ATUISR	32	+078H
ATU Interrupt Mask Register — ATUIMR	32	+07CH
Reserved	32	+080H

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.



Table 667. Address Translation Unit Registers — ATUX (Sheet 2 of 3)

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUX Base Address Offset)
Reserved	32	+084H
Reserved	32	+088H
Reserved	32	+08CH
VPD Capability Identifier Register — VPD_Cap_ID	8	+090H
VPD Next Item Pointer Register — VPD_Next_Item_Ptr	8	+091H
VPD Address Register — VPDAR	16	+092H
VPD Data Register — VPDDR	32	+094H
PM Capability Identifier Register — PM_Cap_ID	8	+098H
PM Next Item Pointer Register — PM_Next_Item_Ptr	8	+099H
ATU Power Management Capabilities Register — APMCR	16	+09AH
ATU Power Management Control/Status Register — APMCSR	16	+09CH
MSI_Capability Identifier Register — MSI_Cap_ID ¹	8	+0A0H
MSI Next Item Pointer Register — MSI_Next_Item_Ptr	8	+0A1H
MSI Message Control Register — MSI_MCR	16	+0A2H
MSI Address Register — MSI_ADDR	32	+0A4H
MSI Message Upper Address Register — MSI_MUAR	32	+0A8H
MSI Message Data Register — MSI_MD	16	+0ACH
Reserved	16	+0AEH
MSI-X Capability Identifier Register — MSI-X_Cap_ID	8	+0B0H
MSI-X Next Item Pointer Register — MSI-X_Next_Item_Ptr	8	+0B1H
MSI-X Message Control Register — MSI-X_MCR	16	+0B2H
MSI-X Table Offset Register — MSI-X_Table_Offset	32	+0B4H
MSI-X Pending Bit Array Offset Register — MSI-X_PBA_Offset	32	+0B8H
MU MSI-X Control Register x — MMCRx	32	+0BCH
Reserved	x	+0C0H through 0CFH
PCI-X Capability Identifier Register — PCI-X_Cap_ID	8	+0D0H
PCI-X Next Item Pointer Register — PCI-X_Next_Item_Ptr	8	+0D1H
PCI-X Command Register — PCI_XCMD	16	+0D2H
PCI-X Status Register — PCI_XSR	32	+0D4H
ECC Control and Status Register — ECCCSR	32	+0D8H
ECC First Address Register — ECCFAR	32	+0DCH
ECC Second Address Register — ECCSAR	32	+0E0H
ECC Attribute Register — ECCAR	32	+0E4H
CompactPCI* Hot-Swap Capability ID Register	8	+0E8H
Offset EDh: HS_NXTP — Next Item Pointer	8	+0E9H
HS_CNTRL — Hot-Swap Control/Status Register	8	+0EAH
Reserved	x	+0EBH through 1FFH
Inbound ATU Base Address Register 3 — IABAR3	32	+200H
Inbound ATU Upper Base Address Register 3 — IAUBAR3	32	+204H
Inbound ATU Limit Register 3 — IALR3	32	+208H
Inbound ATU Translate Value Register 3 — IATVR3	32	+20CH
Inbound ATU Upper Translate Value Register 3 — IAUTVR3	32	+210H
Reserved	x	+214H through 2FFH
Outbound I/O Base Address Register — OIOBAR	32	+300H

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.

**Table 667. Address Translation Unit Registers — ATUX (Sheet 3 of 3)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUX Base Address Offset)
Outbound I/O Window Translate Value Register — OIOWTVR	32	+304H
Outbound Upper Memory Window Base Address Register 0 - OUMBAR0	32	+308H
Outbound Upper 32-bit Memory Window Translate Value Register 0 — OUMWTVR0	32	+30CH
Outbound Upper Memory Window Base Address Register 1 - OUMBAR1	32	+310H
Outbound Upper 32-bit Memory Window Translate Value Register 1 — OUMWTVR1	32	+314H
Outbound Upper Memory Window Base Address Register 2 - OUMBAR2	32	+318H
Outbound Upper 32-bit Memory Window Translate Value Register 2 — OUMWTVR2	32	+31CH
Outbound Upper Memory Window Base Address Register 3 - OUMBAR3	32	+320H
Outbound Upper 32-bit Memory Window Translate Value Register 3 — OUMWTVR3	32	+324H
Reserved.	32	+328H
Reserved.	32	+32CH
Outbound Configuration Cycle Address Register — OCCAR	32	+330H
Outbound Configuration Cycle Data Register — OCCDR	32	+334H
Outbound Configuration Cycle Function Number — OCCFN	32	+338H
Reserved	x	+33CH through +37FH
PCI Interface Error Control and Status Register — PIECSR	32	+380H
PCI Interface Error Address Register — PCIEAR	32	+384H
PCI Interface Error Upper Address Register — PCIEUAR	32	+388H
PCI Interface Error Context Address Register — PCIECAR	32	+38CH
Reserved	x	+390H
Internal Arbiter Control Register — IACR	16	+394H
Reserved	x	+396H
Multi-Transaction Timer — MTT	8	+398H
Reserved	x	+39CH through +FFFH

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.



21.6.2.2 Address Translation Unit (PCI-E)

All of the ATU registers are accessible through both inbound PCI configuration cycles and the 81341 and 81342 core CPU (Register offsets 000H through FFFH).

The Internal Bus Address Offset to PMMRBAR of any ATU Register can be derived by adding the 4 KB address aligned ATUE Base Address Offset (Table 668, “81341 and 81342 ATUE Configuration Space Base Address Offset” on page 1046) to the Register Offset (Table 669, “Address Translation Unit Registers — ATUE” on page 1047).

For example, when INTERFACE_SEL_PCIX# is asserted, the offset to PMMRBAR of the ATU Command Register would be (4 D000H+004H) or 4 D004H.

Note: The 4 KB Address Aligned Range Offset can be different depending on two configuration straps as described in Table 668.

Table 668. 81341 and 81342 ATUE Configuration Space Base Address Offset

INTERFACE_SEL_PCIX#	ATUE Base Address Offset (Relative to PMMRBAR)
PCI Attributes	
Deasserted	+4 0000H
Asserted	+4 5000H
Local Attributes	
Deasserted	+4 8000H
Asserted	+4 D000H

**Table 669. Address Translation Unit Registers — ATUE (Sheet 1 of 4)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUE Base Address Offset)
ATU Vendor ID Register — ATUVID	16	+000H
ATU Device ID Register — ATUDID	16	+002H
ATU Command Register — ATUCMD	16	+004H
ATU Status Register — ATUSR	16	+006H
ATU Revision ID Register — ATURID	8	+008H
ATU Class Code Register — ATUCCR	24	+009H
ATU Cacheline Size Register — ATUCLSR	8	+00CH
ATU Latency Timer Register — ATULT	8	+00DH
ATU Header Type Register — ATUHTR	8	+00EH
ATU BIST Register — ATUBISTR	8	+00FH
Inbound ATU Base Address Register 0 — IABAR0	32	+010H
Inbound ATU Upper Base Address Register 0 — IAUBAR0	32	+014H
Inbound ATU Base Address Register 1 — IABAR1	32	+018H
Inbound ATU Upper Base Address Register 1 — IAUBAR1	32	+01CH
Inbound ATU Base Address Register 2 — IABAR2	32	+020H
Inbound ATU Upper Base Address Register 2 — IAUBAR2	32	+024H
Reserved	32	+028H
ATU Subsystem Vendor ID Register — ASVIR	16	+02CH
ATU Subsystem ID Register — ASIR	16	+02EH
Expansion ROM Base Address Register -ERBAR	32	+030H
ATU Capabilities Pointer Register — ATU_Cap_Ptr	8	+034H
Reserved	24	+035H
Reserved	32	+038H
ATU Interrupt Line Register — ATUILR	8	+03CH
ATU Interrupt Pin Register — ATUIPR	8	+03DH
ATU Minimum Grant Register — ATUMGNT	8	+03EH
ATU Maximum Latency Register — ATUMLAT	8	+03FH
Inbound ATU Limit Register 0 — IALR0	32	+040H
Inbound ATU Translate Value Register 0 — IATVR0	32	+044H
Inbound ATU Upper Translate Value Register 0 — IAUTVR0	32	+048H
Inbound ATU Limit Register 1 — IALR1	32	+04CH
Inbound ATU Translate Value Register 1 — IATVR1	32	+050H
Inbound ATU Upper Translate Value Register 1 — IAUTVR1	32	+054H
Inbound ATU Limit Register 2 — IALR2	32	+058H
Inbound ATU Translate Value Register 2 — IATVR2	32	+05CH
Inbound ATU Upper Translate Value Register 2 — IAUTVR2	32	+060H
Expansion ROM Limit Register — ERLR	32	+064H
Expansion ROM Translate Value Register — ERTVR	32	+068H
Expansion ROM Upper Translate Value Register — ERUTVR	32	+06CH
ATU Configuration Register — ATUCR	32	+070H
PCI Configuration and Status Register — PCSR	32	+074H
ATU Interrupt Status Register — ATUISR	32	+078H
ATU Interrupt Mask Register — ATUIMR	32	+07CH
PCI Express Message Control/Status Register — PEMCSR	32	+080H

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.

**Table 669. Address Translation Unit Registers — ATUE (Sheet 2 of 4)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUE Base Address Offset)
PCI Express Link Control/Status Register — PELCSR	32	+084H
Reserved	x	+088H through +08FH
VPD Capability Identifier Register — VPD_Cap_ID	8	+090H
VPD Next Item Pointer Register — VPD_Next_Item_Ptr	8	+091H
VPD Address Register — VPDAR	16	+092H
VPD Data Register — VPDDR	32	+094H
PM_Capability Identifier Register — PM_Cap_ID	8	+098H
PM Next Item Pointer Register — PM_Next_Item_Ptr	8	+099H
ATU Power Management Capabilities Register — APMCR	16	+09AH
ATU Power Management Control/Status Register — APMCSR	16	+09CH
Reserved	16	+09EH
MSI_Capability Identifier Register — MSI_Cap_ID ¹	8	+0A0H
MSI Next Item Pointer Register — MSI_Next_Item_Ptr	8	+0A1H
MSI Message Control Register — MSI_MCR	16	+0A2H
MSI Address Register — MSI_ADDR	32	+0A4H
MSI Message Upper Address Register — MSI_MUAR	32	+0A8H
MSI Message Data Register — MSI_MD	16	+0ACH
Reserved	16	+0AEH
MSI-X Capability Identifier Register — MSI-X_Cap_ID	8	+0B0H
MSI-X Next Item Pointer Register — MSI-X_Next_Item_Ptr	8	+0B1H
MSI-X Message Control Register — MSI-X_MCR	16	+0B2H
MSI-X Table Offset Register — MSI-X_Table_Offset	32	+0B4H
MSI-X Pending Bit Array Offset Register — MSI-X_PBA_Offset	32	+0B8H
MU MSI-X Control Register x — MMCRx	32	+0BCH
Reserved	x	+0C0H through +0CFH
PCI Express Capability Identifier Register — PCIE_CAPID	8	+0D0H
PCI Express Next Item Pointer Register — PCIE_NXTP	8	+0D1H
PCI Express Capabilities Register PCIE_CAP	16	+0D2H
PCI Express Device Capabilities Register — PCIE_DCAP	32	+0D4H
PCI Express Device Control Register — PE_DCTL	16	+0D8H
PCI Express Device Status Register PE_DSTS	16	+0DAH
PCI Express Link Capabilities Register — PE_LCAP	32	+0DCH
PCI Express Link Control Register PE_LCTL	16	+0E0H
PCI Express Link Status Register PE_LSTS	16	+0E2H
PCI Express Slot Capabilities Register — PE_SCAP	32	+0E4H
PCI Express Slot Control Register PE_SCR	16	+0E8H
PCI Express Slot Status Register PE_SSTS	16	+0EAH
PCI Express Root Control Register — PE_RCR	16	+0ECH
Reserved	16	+0EEH
PCI Express Root Status Register PE_RSR	32	+0F0H
Reserved	96	+0F4H through +0FFH
PCI Express Advanced Error Capability Identifier — ADVERR_CAPID	32	+100H
PCI Express Uncorrectable Error Status — ERRUNC_STS	32	+104H
PCI Express Uncorrectable Error Mask — ERRUNC_MSK	32	+108H

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.

**Table 669. Address Translation Unit Registers — ATUE (Sheet 3 of 4)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUE Base Address Offset)
PCI Express Uncorrectable Error Severity — ERRUNC_SEV	32	+10CH
PCI Express Correctable Error Status — ERRCOR_STS	32	+110H
PCI Express Correctable Error Mask — ERRCOR_MSK	32	+114H
Advanced Error Control and Capability Register — ADVERR_CTL	32	+118H
PCI Express Advanced Error Header Log — ADVERR_LOG0	32	+11CH
PCI Express Advanced Error Header Log — ADVERR_LOG1	32	+120H
PCI Express Advanced Error Header Log — ADVERR_LOG2	32	+124H
PCI Express Advanced Error Header Log — ADVERR_LOG3	32	+128H
Root Error Command Register — RERR_CMD	32	+12CH
Root Error Status Register — RERR_SR	32	+130H
Error Source Identification Register RERR_ID	32	+134H
Reserved.	32	+140H
Reserved.	32	+144H
Reserved.	32	+148H
Reserved.	32	+14CH
Reserved.	x	+150H through +1DFH
Device Serial Number Capability — DSN_CAP	32	+1E0H
Device Serial Number Lower DW Register — DSN_LDW	32	+1E4H
Device Serial Number Upper DW Register — DSN_UDW	32	+1E8H
Power Budgeting Enhanced Capability Header — PWRBGT_CAPID	32	+1F0H
Power Budgeting Data Select Register — PWRBGT_DSEL	32	+1F4H
Power Budgeting Data Register — PWRBGT_DATA	32	+1F8H
Power Budgeting Capability Register — PWRBGT_CAP	32	+1FCH
Power Budgeting Information Registers[0:23] — PWRBGT_INFO[0:23]	32 x24	+200H through +25CH
Reserved.	x	+260H through +2FFH
Outbound I/O Base Address Register — OIOBAR	32	+300H
Outbound I/O Window Translate Value Register — OIOWTVR	32	+304H
Outbound Upper Memory Window Base Address Register 0 — OUMBAR0	32	+308H
Outbound Upper 32-bit Memory Window Translate Value Register 0- OUMWTVR0	32	+30CH
Outbound Upper Memory Window Base Address Register 1 — OUMBAR1	32	+310H
Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1	32	+314H
Outbound Upper Memory Window Base Address Register 2- OUMBAR2	32	+318H
Outbound Upper 32-bit Memory Window Translate Value Register 2- OUMWTVR2	32	+31CH
Outbound Upper Memory Window Base Address Register 3 — OUMBAR3	32	+320H
Outbound Upper 32-bit Memory Window Translate Value Register 3- OUMWTVR3	32	+324H
Reserved.	32	+328H
Outbound Configuration Cycle Address Register — OCCAR	32	+32CH
Outbound Configuration Cycle Data Register — OCCDR	32	+330H
Outbound Configuration Cycle Function Number — OCCFN	32	+334H
Reserved.	x	+338H through +33FH
Inbound Vendor Defined Message Header Register0 — IVMHR0	32	+340H
Inbound Vendor Defined Message Header Register 1 — IVMHR1	32	+344H
Inbound Vendor Defined Message Header Register 2 — IVMHR2	32	+348H
Inbound Vendor Defined Message Header Register 3 — IVMHR3	32	+34CH

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.

**Table 669. Address Translation Unit Registers — ATUE (Sheet 4 of 4)**

Register Description (Name)	Register Size in Bits	Internal Bus Address Offset (Relative to ATUE Base Address Offset)
Inbound Vendor Defined Message Payload Register — IVMPR	32	+350H
Reserved.	x	+354H through +35FH
Outbound Vendor Defined Message Header Register0 — OVMHR0	32	+360H
Outbound Vendor Defined Message Header Register 1 — OVMHR1	32	+364H
Outbound Vendor Defined Message Header Register 2 — OVMHR2	32	+368H
Outbound Vendor Defined Message Header Register 3 — OVMHR3	32	+36CH
Outbound Vendor Defined Message Payload Register — OVMPR	32	+370H
Reserved.	x	+374H through +37FH
PCI Interface Error Control and Status Register — PIE_CSR	32	+380H
PCI Interface Error Status — PIE_STS	32	+384H
PCI Interface Error Mask — PIE_MSK	32	+388H
PCI Interface Error Header Log — PIE_LOG0	32	+38CH
PCI Interface Error Header Log 1 — PIE_LOG1	32	+390H
PCI Interface Error Header Log 2 — PIE_LOG2	32	+394H
PCI Interface Error Header Log — PIE_LOG3	32	+398H
PCI Interface Error Header Log — PIE_DLOG	32	+39CH
Reserved.	x	+3A0H through +FFFH

Notes:

1. MSI and MSI-X Capability Registers are documented in the Messaging Unit Chapter.



21.7 PCI Configuration Space

The PCI Configuration space of the 81341 and 81342 varies depending on the INTERFACE_SEL_PCIX# strap. The PCI Functions that are visible in via configuration transactions are details in [Table 670, “Intel® 81348 I/O Processor PCI Function Visibility”](#).

Configurations cycles that target 81341 and 81342 are translated into memory cycles on the internal bus and access the PCI Attributes section of the PMMR region. The internal register locations are described in [Section 21.6.2, “Host Interface Units”](#) on [page 1041](#).

Table 670. Intel® 81348 I/O Processor PCI Function Visibility

	PCI Function Number							
	0	1	2	3	4	5	6	7
XX	ATUX ^a ATUE ^b	Reserved						

a. ATUX is visible in function 0 if the INTERFACE_SEL_PCIX# strap is sampled as 0.

b. ATUE is visible in function 0 if the INTERFACE_SEL_PCIX# strap is sampled as 1.



21.8 Coprocessor Register Space

The CCR address space is assigned to support the integrated peripherals on the 81341 and 81342 that require low latency register access. Table 671 shows all of the 81341 and 81342 integrated coprocessor registers and assigned coprocessor space. The *ARM Architecture Reference Manual* provides for a total of 16 coprocessors each of which can contain up to 256 32 bit registers. For completeness, the coprocessor space reserved by the *ARM Architecture Reference Manual* is shown.

Note: All accesses to CP6 unimplemented coprocessor registers complete and return 0s when read and show as “undefined”. The same rule applies to unimplemented 81341 and 81342. co-processors.

Table 671. Coprocessor Registers Assigned to Integrated Peripherals

Integrated Peripheral	Coprocessor
Inter-Processor Communication Unit	CP6
Interrupt Control Unit	CP6
Programmable Timers	CP6
Bus Interface Unit	CP7
Core Performance Monitoring Unit	CP14
System Control ^a	CP15

a. Reserved by the *ARM Architecture Reference Manual*.

**Table 672. Coprocessor Register Locations (Sheet 1 of 4)**

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
Inter-Processor Communication	Core Identification Register — CIDR	CP6	0	Register 0
	Reset Cause Status Register — RCSR		1	Register 0
	Software Interrupt Generation Register — SINTGENR			Register 1
	Targeted Reset Register — TARRSTR			Register 2
	Undefined			Register 3 through Register 7
	Inbound MSI Interrupt Pending Register 0			Register 8
	Inbound MSI Interrupt Pending Register 1			Register 9
	Inbound MSI Interrupt Pending Register 2			Register 10
	Inbound MSI Interrupt Pending Register 3			Register 11
	Undefined			Register 12 through Register 15
Interrupt Control Unit	Interrupt Base Register	CP6	2	Register 0
	Undefined			Register 1
	Interrupt Size Register			Register 2
	IRQ Interrupt Vector Register			Register 3
	FIQ Interrupt Vector Register			Register 4
	Undefined			Register 4–7
	Inter-Processor Interrupt Pending Register			Register 8
	Undefined			Register 9–15
	Interrupt Pending Register 0		3	Register 0
	Interrupt Pending Register 1			Register 1
	Interrupt Pending Register 2			Register 2
	Interrupt Pending Register 3			Register 3
	Undefined			Register 4–15
	Interrupt Control Register 0		4	Register 0
	Interrupt Control Register 1			Register 1
	Interrupt Control Register 2			Register 2
	Interrupt Control Register 3			Register 3
	Undefined			Register 4–15



Table 672. Coprocessor Register Locations (Sheet 2 of 4)

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
	Interrupt Steering Register 0		5	Register 0
	Interrupt Steering Register 1			Register 1
	Interrupt Steering Register 2			Register 2
	Interrupt Steering Register 3			Register 3
	Undefined			Register 4–15
Interrupt Control Unit	IRQ Interrupt Source Register 0	CP6	6	Register 0
	IRQ Interrupt Source Register 1			Register 1
	IRQ Interrupt Source Register 2			Register 2
	IRQ Interrupt Source Register 3			Register 3
	Undefined			Register 4–15
	FIQ Interrupt Source Register 0		7	Register 0
	FIQ Interrupt Source Register 1			Register 1
	FIQ Interrupt Source Register 2			Register 2
	FIQ Interrupt Source Register 3			Register 3
	Undefined			Register 4–15
	Interrupt Priority Register 0		8	Register 0
	Interrupt Priority Register 1			Register 1
	Interrupt Priority Register 2			Register 2
	Interrupt Priority Register 3			Register 3
	Interrupt Priority Register 4			Register 4
	Interrupt Priority Register 5			Register 5
	Interrupt Priority Register 6			Register 6
	Interrupt Priority Register 7			Register 7
	Undefined			Register 8–15

**Table 672. Coprocessor Register Locations (Sheet 3 of 4)**

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
Programmable Timers Unit	Timer Mode Register 0	CP6	9	Register 0
	Timer Mode Register 1			Register 1
	Timer Count Register 0			Register 2
	Timer Count Register 1			Register 3
	Timer Reload Register 0			Register 4
	Timer Reload Register 1			Register 5
	Timer Interrupt Status Register			Register 6
	Watch Dog Timer Control Register			Register 7
	Watch Dog Timer Setup Register			Register 8
	Undefined			Register 9–15
Bus Interface Unit	L2 Cache and BIU Error Logging Register	CP7	2	Register 0
	L2 Cache and BIU Error Lower Address Register			Register 1
	L2 Cache and BIU Error Upper Address Register			Register 2
	Undefined			Register 3–15
Clock and Power Management Unit	Undefined	CP14	0	Register 0–5
	Core Clock Configuration Register			Register 6
	Power Mode Register			Register 7
	Transmit Register			Register 8
	Receive Register			Register 9
	Debug Control and Status Register			Register 10
	Trace Buffer Register			Register 11
	Checkpoint 0 Register			Register 12
	Checkpoint 1 Register			Register 13
	Transmit/Receive Control Register			Register 14
	Undefined			Register 15



Table 672. Coprocessor Register Locations (Sheet 4 of 4)

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
Core Performance Monitoring Unit	Performance Monitor Control Register	CP14	1	Register 0
	Clock Count Register			Register 1
	Undefined			Register 2
	Undefined			Register 3
	Interrupt Enable Register			Register 4
	Overflow Flag Register			Register 5
	Undefined			Register 6
	Undefined			Register 7
	Event Selection Register			Register 8
	Undefined			Register 9–15
	Performance Count Register 0	CP14	2	Register 0
	Performance Count Register 1			Register 1
	Performance Count Register 2			Register 2
	Performance Count Register 3			Register 3
	Undefined			Register 4–15
System Controls	ID & Cache Type Registers	CP15	CP 15 Function ^a	Register 0
	Control and Auxiliary Control Registers			Register 1
	Translation Table Base Register			Register 2
	Domain Access Control Register			Register 3
	Undefined			Register 4
	Fault Status Register			Register 5
	Fault Address Register			Register 6
	Cache Functions Register			Register 7
	TLB Operations Register			Register 8
	Cache Lock Down			Register 9
	TLB Lock Down			Register 10
	Reserved			Register 11–12
	Process ID Register			Register 13
	Breakpoint Registers			Register 14
	CP Access (PID) Coprocessor Access Control Register			Register 15

a. Some of the CP15 registers are differentiated by the Opcode_2 field. For CP15, the CR_m is used in some cases to denote different control functions for a given coprocessor register rather than a distinct register decode. Please refer to the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual* (Order Number: 273411), for more details on the operation of CP15.

