Processor Architecture

2015.04.02

NII51002

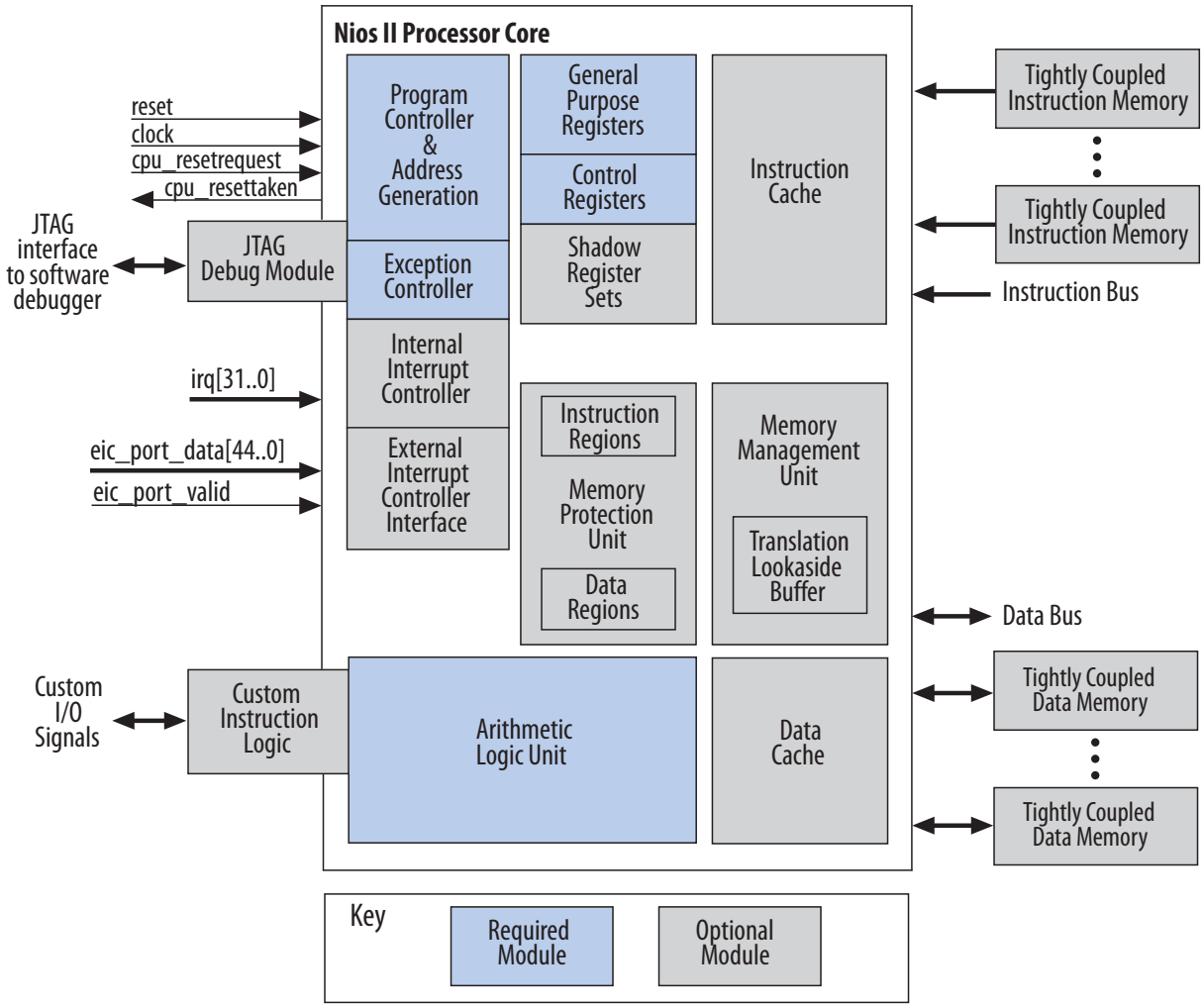
This chapter describes the hardware structure of the Nios II processor, including a discussion of all the functional units of the Nios II architecture and the fundamentals of the Nios II processor hardware implementation.

The Nios II architecture describes an instruction set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions. A Nios II processor core is a hardware design that implements the Nios II instruction set and supports the functional units described in this document. The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture.

The Nios II architecture defines the following functional units:

- Register file
- Arithmetic logic unit (ALU)
- Interface to custom instruction logic
- Exception controller
- Internal or external interrupt controller
- Instruction bus
- Data bus
- Memory management unit (MMU)
- Memory protection unit (MPU)
- Instruction and data cache memories
- Tightly-coupled memory interfaces for instructions and data
- JTAG debug module

Figure 1: Nios II Processor Core Block Diagram



Processor Implementation

The functional units of the Nios II architecture form the foundation for the Nios II instruction set. However, this does not indicate that any unit is implemented in hardware. The Nios II architecture describes an instruction set, not a particular hardware implementation. A functional unit can be implemented in hardware, emulated in software, or omitted entirely.

A Nios II implementation is a set of design choices embodied by a particular Nios II processor core. All implementations support the instruction set defined in the *Instruction Set Reference* chapter.

Each implementation achieves specific objectives, such as smaller core size or higher performance. This flexibility allows the Nios II architecture to adapt to different target applications.

Implementation variables generally fit one of three trade-off patterns: more or less of a feature; inclusion or exclusion of a feature; hardware implementation or software emulation of a feature. An example of each trade-off follows:

- More or less of a feature—For example, to fine-tune performance, you can increase or decrease the amount of instruction cache memory. A larger cache increases execution speed of large programs, while a smaller cache conserves on-chip memory resources.
- Inclusion or exclusion of a feature—For example, to reduce cost, you can choose to omit the JTAG debug module. This decision conserves on-chip logic and memory resources, but it eliminates the ability to use a software debugger to debug applications.
- Hardware implementation or software emulation—For example, in control applications that rarely perform complex arithmetic, you can choose for the division instruction to be emulated in software. Removing the divide hardware conserves on-chip resources but increases the execution time of division operations.

For information about which Nios II cores supports what features, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

For complete details about user-selectable parameters for the Nios II processor, refer to the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook*.

Related Information

- **Instantiating the Nios II Processor**
- **Nios II Core Implementation Details**
- **Instruction Set Reference**

Register File

The Nios II architecture supports a flat register file, consisting of thirty-two 32-bit general-purpose integer registers, and up to thirty-two 32-bit control registers. The architecture supports supervisor and user modes that allow system code to protect the control registers from errant applications.

The Nios II processor can optionally have one or more shadow register sets. A shadow register set is a complete set of Nios II general-purpose registers. When shadow register sets are implemented, the `CRS` field of the `status` register indicates which register set is currently in use. An instruction access to a general-purpose register uses whichever register set is active.

A typical use of shadow register sets is to accelerate context switching. When shadow register sets are implemented, the Nios II processor has two special instructions, `rdprs` and `wrprs`, for moving data between register sets. Shadow register sets are typically manipulated by an operating system kernel, and are transparent to application code. A Nios II processor can have up to 63 shadow register sets.

The Nios II architecture allows for the future addition of floating-point registers.

For details about shadow register set implementation and usage, refer to “Registers” and “Exception Processing” in the *Programming Model* chapter of the Nios II Processor Reference Handbook.

For details about the `rdprs` and `wrprs` instructions, refer to the *Instruction Set Reference* chapter of the *Nios II Processor Reference Handbook*.

Related Information

- **Programming Model**
- **Instruction Set Reference**

Arithmetic Logic Unit

The Nios II ALU operates on data stored in general-purpose registers. ALU operations take one or two inputs from registers, and store a result back in a register. The ALU supports the data operations described in the table below. To implement any other operation, software computes the result by performing a combination of the fundamental operations.

Table 1: Operations Supported by the Nios II ALU

Category	Details
Arithmetic	The ALU supports addition, subtraction, multiplication, and division on signed and unsigned operands.
Relational	The ALU supports the equal, not-equal, greater-than-or-equal, and less-than relational operations (==, != >=, <) on signed and unsigned operands.
Logical	The ALU supports AND, OR, NOR, and XOR logical operations.
Shift and Rotate	The ALU supports shift and rotate operations, and can shift/rotate data by 0 to 31 bit positions per instruction. The ALU supports arithmetic shift right and logical shift right/left. The ALU supports rotate left/right.

Unimplemented Instructions

Some Nios II processor core implementations do not provide hardware to support the entire Nios II instruction set. In such a core, instructions without hardware support are known as unimplemented instructions.

The processor generates an exception whenever it issues an unimplemented instruction so your exception handler can call a routine that emulates the operation in software. Unimplemented instructions do not affect the programmer's view of the processor.

For a list of potential unimplemented instructions, refer to the *Programming Model* chapter of the Nios II Processor Reference Handbook.

Related Information

Programming Model

Custom Instructions

The Nios II architecture supports user-defined custom instructions. The Nios II ALU connects directly to custom instruction logic, enabling you to implement operations in hardware that are accessed and used exactly like native instructions.

Refer to "Custom Instruction Tab" in the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook* for additional information.

Related Information

- **Instantiating the Nios II Processor**
- **Nios II Custom Instruction User Guide**
  For more information, refer to the *Nios II Custom Instruction User Guide*.

Floating-Point Instructions

The Nios II architecture supports single precision floating-point instructions with two components:

- Floating Point Hardware 2—This component supports floating-point instructions as specified by the IEEE Std 754-2008 but with simplified, non-standard rounding modes. The basic set of floating-point custom instructions includes single precision floating-point addition, subtraction, multiplication, division, square root, integer to float conversion, float to integer conversion, minimum, maximum, negate, absolute, and comparisons.
- Floating Point Hardware—This component supports floating-point instructions as specified by the IEEE Std 754-1985. The basic set of floating-point custom instructions includes single precision floating-point addition, subtraction, and multiplication. Floating-point division is available as an extension to the basic instruction set.

These floating-point instructions are implemented as custom instructions. The Hardware Conformance table below lists a detailed description of the conformance to the IEEE standards.

Table 2: Hardware Conformance with IEEE 754-1985 and IEEE 754-2008 Floating-Point Standard

Feature		Floating-Point Hardware Implementation with IEEE 754-1985	Floating-Point Hardware 2 Implementation with IEEE 754-2008
Operations	Addition/subtraction	Implemented	Implemented
	Multiplication	Implemented	Implemented
	Division	Optional	Implemented
	Square root	Not implemented, this operation is implemented in software.	Implemented
	Integer to float/float to integer	Not implemented, this operation is implemented in software.	Implemented
	Minimum/maximum	Not implemented, this operation is implemented in software.	Implemented
	Negate/absolute	Not implemented, this operation is implemented in software.	Implemented
	Comparisons	Not implemented, this operation is implemented in software.	Implemented
Precision	Single	Implemented	Implemented
	Double	Not implemented. Double precision operations are implemented in software.	Not implemented. Double precision operations are implemented in software.

Feature		Floating-Point Hardware Implementation with IEEE 754-1985	Floating-Point Hardware 2 Implementation with IEEE 754-2008
Exception conditions	Invalid operation	Result is Not a Number (NaN)	Result is Not a Number (NaN)
	Division by zero	Result is ±infinity	Result is ±infinity
	Overflow	Result is ±infinity	Result is ±infinity
	Inexact	Result is a normal number	Result is a normal number
	Underflow	Result is ±0	Result is ±0
Rounding Modes	Round to nearest	Implemented	Implemented (roundTies-ToAway mode)
	Round toward zero	Not implemented	Implemented (truncation mode)
	Round toward +infinity	Not implemented	Not implemented
	Round toward –infinity	Not implemented	Not implemented
NaN	Quiet	Implemented	No distinction is made between signaling and quiet NaNs as input operands. A result that produces a NaN may produce either a signaling or quiet NaN.(1)
	Signaling	Not implemented	
Subnormal (denormalized) numbers		Subnormal operands are treated as zero. The floating-point custom instructions do not generate subnormal numbers.	• The comparison, minimum, maximum, negate, and absolute operations support subnormal numbers. • The add, subtract, multiply, divide, square root, and float to integer operations do NOT support subnormal numbers. Subnormal operands are treated as signed zero. The floating-point custom instructions do not generate subnormal numbers.(1) • The integer to float operation cannot create subnormal numbers.

(1) This operation is not fully compliant with IEEE 754-2008.

Feature		Floating-Point Hardware Implementation with IEEE 754-1985	Floating-Point Hardware 2 Implementation with IEEE 754-2008
Software exceptions		Not implemented. IEEE 754-1985 exception conditions are detected and handled as described elsewhere in this table.	Not implemented. IEEE 754-2008 exception conditions are detected and handled as described elsewhere in this table.(1)
Status flags		Not implemented. IEEE 754-1985 exception conditions are detected and handled as described elsewhere in this table.	Not implemented. IEEE 754-2008 exception conditions are detected and handled as described elsewhere in this table.(1)

Note: The Floating Point Hardware 2 component also supports faithful rounding, which is not an IEEE 754-defined rounding mode. Faithful rounding rounds results to either the upper or lower nearest single-precision numbers. Therefore, the result produced is one of two possible values and the choice between the two is not defined. The maximum error of faithful rounding is 1 unit in the last place (ulp). Errors may not be evenly distributed.

Related Information

Nios II Custom Instruction User Guide

For more information about using floating-point custom instructions in software, refer to the *Nios II Custom Instruction User Guide*.

Floating Point Custom Instruction 2 Component

You can add floating-point custom instructions to any Nios II processor design. The floating-point division hardware requires more resources than the other instructions. The Floating Point Hardware 2 component supports the following single-precision floating-point operations:

- Add
- Subtract
- Multiply
- Divide
- Square root
- Comparison
- Integer conversion
- Minimum
- Maximum
- Negate
- Absolute

Other floating-point operations (including double-precision operations) are implemented with software emulation. The component requires the following device resources:

- ~2,500 4-input LEs
- 9 x 9 bit multipliers
- 3x M9K memories

In the following table, a and b are assumed to be single-precision floating-point values.

Table 3: Floating Point Custom Instruction 2 Operation Summary

Operation[2]	N[3]	Cycles	Result	Subnormal	Rounding	GCC Inference
fdivs	255	16	a ÷ b	Flush to 0	Nearest	a / b
fsubs	254	5	a – b	Flush to 0	Faithful	a – b
fadds	253	5	a + b	Flush to 0	Faithful	a + b
fmuls	252	4	a x b	Flush to 0	Faithful	a * b
fsqrts	251	8	$\sqrt{a}$	Flush to 0	Faithful	sqrtf()[4]
floatis	250	4	int_to_float(a)	Not applicable	Not applicable	Casting
fixsi	249	2	float_to_int(a)	Flush to 0	Truncation	Casting
round	248	2	float_to_int(a)	Flush to 0	Nearest	lroundf()[4]
Reserved	234 to 247	Undefined	Undefined			
fmins	233	1	(a < b) ? a : b	Supported	None	fminf()[4]
fmaxs	232	1	(a < b) ? b : a	Supported	None	fmaxf()[4]
fcmplts	231	1	(a < b) ? 1 : 0	Supported	None	a < b
fcmples	230	1	(a ≤ b) ? 1 : 0	Supported	None	a <= b
fcmpgts	229	1	(a > b) ? 1 : 0	Supported	None	a > b
fcmpges	228	1	(a ≥ b) ? 1 : 0	Supported	None	a >= b
fcmpeqs	227	1	(a = b) ? 1 : 0	Supported	None	a == b
fcmpnes	226	1	(a ≠ b) ? 1 : 0	Supported	None	a != b
fnegs	225	1	-a	Supported	None	-a
fabss	224	1	|a|	Supported	None	fabsf()

The cycles column specifies the number of cycles required to execute the instruction. A combinatorial custom instruction takes 1 cycle. A multi-cycle custom instruction requires at least 2 cycles. An N-cycle multi-cycle custom instruction has N - 2 register stages inside the custom instruction because the Nios II processor registers the result from the custom instruction and allows another cycle for g wire delays in the source operand bypass multiplexers. The number of cycles does not include the extra cycles (maximum of 2) that an instruction following the multi-cycle custom instruction is stalled by the Nios II/f if the instruc-

(2) These names match the names of the corresponding GCC command-line options except for round, which GCC does not support.
(3) Specifies the 8 bit fixed custom instruction for the operation.
(4) Nios II GCC version 4.7.3 is not able to reliably replace calls to newlib floating-point functions with the equivalent custom instruction even though it has `-mcustom-<operation>` command-line options and pragma support for these operations. Instead, the custom instruction must be invoked directly using the GCC `__builtin_custom_*` facility. The Floating Point Custom Instruction 2 component includes a C header file that provides the required `#define` macros to invoke the custom instruction directly.

tion uses the result within 2 cycles. These extra cycles occur because multi-cycle instructions are late result instructions

In Qsys, the **Floating Point Hardware 2** component is under **Embedded Processors** on the **Component Library** tab.

The Nios II Software Build Tools (SBT) include software support for the Floating Point Custom Instruction 2 component. When the Floating Point Custom Instruction 2 component is present in hardware, the Nios II compiler compiles the software codes to use the custom instructions for floating point operations.

Floating Point Custom Instruction Component

The Floating Point Hardware component supports addition, subtraction, multiplication, and (optionally) division. The Floating Point Hardware parameter editor allows you to omit the floating-point division hardware for cases in which code running on your hardware design does not make heavy use of floating-point division. When you omit the floating-point divide instruction, the Nios II compiler implements floating-point division in software.

In Qsys, the **Floating Point Hardware** component is under **Embedded Processors** on the **Component Library** tab.

The Nios II floating-point custom instructions are based on the Altera® floating-point megafunctions: ALTFP_MULT, ALTFP_ADD_SUB, and ALTFP_DIV.

The Nios II software development tools recognize C code that takes advantage of the floating-point instructions present in the processor core. When the floating-point custom instructions are present in your target hardware, the Nios II compiler compiles your code to use the custom instructions for floating-point operations and the newlib math library.

Related Information

IP and Megafunctions

For information about each individual floating-point megafunction, including acceleration factors and device resource usage, refer to the megafunction user guides, available on the IP and Megafunctions literature page of the Altera website.

Reset and Debug Signals

The table below describes the reset and debug signals that the Nios II processor core supports.

Table 4: Nios II Processor Debug and Reset Signals

Signal Name	Type	Purpose
`reset`	Reset	This is a global hardware reset signal that forces the processor core to reset immediately.

Signal Name	Type	Purpose
cpu_resetrequest	Reset	This is an optional, local reset signal that causes the processor to reset without affecting other components in the Nios II system. The processor finishes executing any instructions in the pipeline, and then enters the reset state. This process can take several clock cycles, so be sure to continue asserting the cpu_resetrequest signal until the processor core asserts a cpu_resettaken signal. The processor core asserts a cpu_resettaken signal for 1 cycle when the reset is complete and then periodically if cpu_resetrequest remains asserted. The processor remains in the reset state for as long as cpu_resetrequest is asserted. While the processor is in the reset state, it periodically reads from the reset address. It discards the result of the read, and remains in the reset state. The processor does not respond to cpu_resetrequest when the processor is under the control of the JTAG debug module, that is, when the processor is paused. The processor responds to the cpu_resetrequest signal if the signal is asserted when the JTAG debug module relinquishes control, both momentarily during each single step as well as when you resume execution.
debugreq	Debug	This is an optional signal that temporarily suspends the processor for debugging purposes. When you assert the signal, the processor pauses in the same manner as when a breakpoint is encountered, transfers execution to the routine located at the break address, and asserts a debugack signal. Asserting the debugreq signal when the processor is already paused has no effect.
reset_req	Reset	This optional signal prevents the memory corruption by performing a reset handshake before the processor resets.

For more information on adding reset signals to the Nios II processor, refer to "Advanced Features Tab" in the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook*.

For more information on the break vector and adding debug signals to the Nios II processor, refer to "JTAG Debug Module Tab" in the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook*.

Related Information

Instantiating the Nios II Processor

Exception and Interrupt Controllers

The Nios II processor includes hardware for handling exceptions, including hardware interrupts. It also includes an optional external interrupt controller (EIC) interface. The EIC interface enables you to speed up interrupt handling in a complex system by adding a custom interrupt controller.

Exception Controller

The Nios II architecture provides a simple, nonvectored exception controller to handle all exception types. Each exception, including internal hardware interrupts, causes the processor to transfer execution to an

exception address. An exception handler at this address determines the cause of the exception and dispatches an appropriate exception routine.

Exception addresses are specified with the Qsys Nios II Processor parameter editor.

All exceptions are precise. Precise means that the processor has completed execution of all instructions preceding the faulting instruction and not started execution of instructions following the faulting instruction. Precise exceptions allow the processor to resume program execution once the exception handler clears the exception.

EIC Interface

An EIC provides high performance hardware interrupts to reduce your program's interrupt latency. An EIC is typically used in conjunction with shadow register sets and when you need more than the 32 interrupts provided by the Nios II internal interrupt controller.

The Nios II processor connects to an EIC through the EIC interface. When an EIC is present, the internal interrupt controller is not implemented; Qsys connects interrupts to the EIC.

The EIC selects among active interrupts and presents one interrupt to the Nios II processor, with interrupt handler address and register set selection information. The interrupt selection algorithm is specific to the EIC implementation, and is typically based on interrupt priorities. The Nios II processor does not depend on any specific interrupt prioritization scheme in the EIC.

For every external interrupt, the EIC presents an interrupt level. The Nios II processor uses the interrupt level in determining when to service the interrupt.

Any external interrupt can be configured as an NMI. NMIs are not masked by the `status.PIE` bit, and have no interrupt level.

An EIC can be software-configurable.

Note: When the EIC interface and shadow register sets are implemented on the Nios II core, you must ensure that your software is built with the Nios II EDS version 9.0 or higher. Earlier versions have an implementation of the `eret` instruction that is incompatible with shadow register sets.

For a typical example of an EIC, refer to the *Vectored Interrupt Controller* chapter in the Embedded Peripherals IP User Guide.

For details about EIC usage, refer to "Exception Processing" in the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Related Information

- **Embedded Peripherals IP User Guide**
  For a typical example of an EIC, refer to the *Vectored Interrupt Controller* chapter in the *Embedded Peripherals IP User Guide*.
- **Programming Model**

Internal Interrupt Controller

The Nios II architecture supports 32 internal hardware interrupts. The processor core has 32 level-sensitive interrupt request (IRQ) inputs, `irq0` through `irq31`, providing a unique input for each interrupt source. IRQ priority is determined by software. The architecture supports nested interrupts.

Your software can enable and disable any interrupt source individually through the `ienable` control register, which contains an interrupt-enable bit for each of the IRQ inputs. Software can enable and

disable interrupts globally using the PIE bit of the `status` control register. A hardware interrupt is generated if and only if all of the following conditions are true:

- The PIE bit of the `status` register is 1
- An interrupt-request input, `irq<n>`, is asserted
- The corresponding bit n of the `ienable` register is 1

The interrupt vector custom instruction is less efficient than using the EIC interface with the Altera vectored interrupt controller component, and thus is deprecated in Qsys. Altera recommends using the EIC interface.

Memory and I/O Organization

This section explains hardware implementation details of the Nios II memory and I/O organization. The discussion covers both general concepts true of all Nios II processor systems, as well as features that might change from system to system.

The flexible nature of the Nios II memory and I/O organization are the most notable difference between Nios II processor systems and traditional microcontrollers. Because Nios II processor systems are configurable, the memories and peripherals vary from system to system. As a result, the memory and I/O organization varies from system to system.

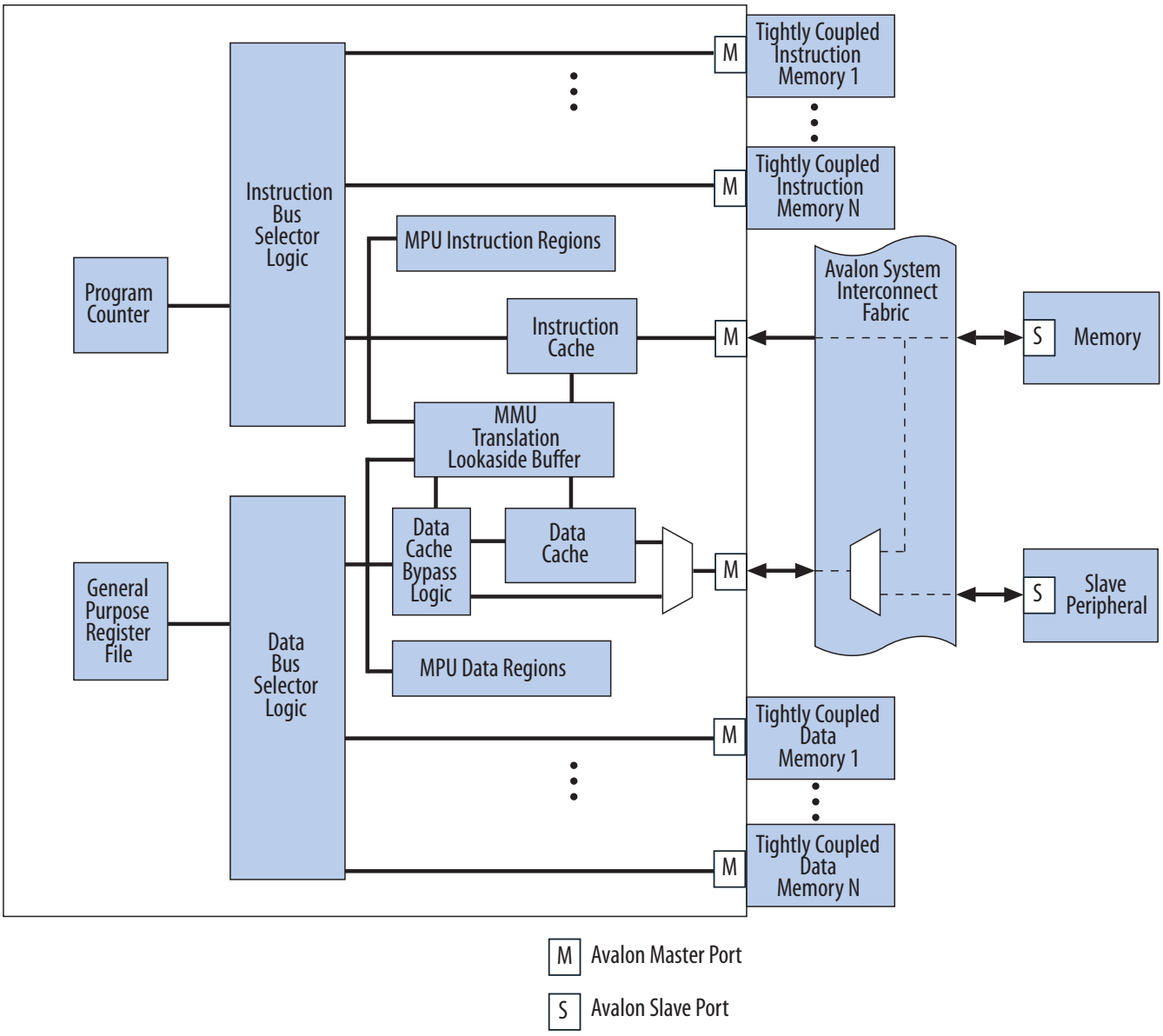
A Nios II core uses one or more of the following to provide memory and I/O access:

- Instruction master port—An Avalon® Memory-Mapped (Avalon-MM) master port that connects to instruction memory via system interconnect fabric
- Instruction cache—Fast cache memory internal to the Nios II core
- Data master port—An Avalon-MM master port that connects to data memory and peripherals via system interconnect fabric
- Data cache—Fast cache memory internal to the Nios II core
- Tightly-coupled instruction or data memory port—Interface to fast on-chip memory outside the Nios II core

The Nios II architecture handles the hardware details for the programmer, so programmers can develop Nios II applications without specific knowledge of the hardware implementation.

For details that affect programming issues, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Figure 2: Nios II Memory and I/O Organization



Related Information

Programming Model

Instruction and Data Buses

The Nios II architecture supports separate instruction and data buses, classifying it as a Harvard architecture. Both the instruction and data buses are implemented as Avalon-MM master ports that adhere to the Avalon-MM interface specification. The data master port connects to both memory and peripheral components, while the instruction master port connects only to memory components.

Related Information

Avalon Interface Specifications
Refer to the Avalon Interface Specifications for details of the Avalon-MM interface.

Memory and Peripheral Access

The Nios II architecture provides memory-mapped I/O access. Both data memory and peripherals are mapped into the address space of the data master port. The Nios II architecture uses little-endian byte ordering. Words and halfwords are stored in memory with the more-significant bytes at higher addresses.

The Nios II architecture does not specify anything about the existence of memory and peripherals; the quantity, type, and connection of memory and peripherals are system-dependent. Typically, Nios II processor systems contain a mix of fast on-chip memory and slower off-chip memory. Peripherals typically reside on-chip, although interfaces to off-chip peripherals also exist.

Instruction Master Port

The Nios II instruction bus is implemented as a 32-bit Avalon-MM master port. The instruction master port performs a single function: it fetches instructions to be executed by the processor. The instruction master port does not perform any write operations.

The instruction master port is a pipelined Avalon-MM master port. Support for pipelined Avalon-MM transfers minimizes the impact of synchronous memory with pipeline latency and increases the overall f_{MAX} of the system. The instruction master port can issue successive read requests before data has returned from prior requests. The Nios II processor can prefetch sequential instructions and perform branch prediction to keep the instruction pipe as active as possible.

The instruction master port always retrieves 32 bits of data. The instruction master port relies on dynamic bus-sizing logic contained in the system interconnect fabric. By virtue of dynamic bus sizing, every instruction fetch returns a full instruction word, regardless of the width of the target memory. Consequently, programs do not need to be aware of the widths of memory in the Nios II processor system.

The Nios II architecture supports on-chip cache memory for improving average instruction fetch performance when accessing slower memory. Refer to the "Cache Memory" section of this chapter for details.

The Nios II architecture supports tightly-coupled memory, which provides guaranteed low-latency access to on-chip memory. Refer to the "Tightly-Coupled Memory" section of this chapter for details.

Related Information

- **Cache Memory** on page 15
- **Tightly-Coupled Memory** on page 16

Data Master Port

The Nios II data bus is implemented as a 32-bit Avalon-MM master port. The data master port performs two functions:

- Read data from memory or a peripheral when the processor executes a load instruction
- Write data to memory or a peripheral when the processor executes a store instruction

Byte-enable signals on the master port specify which of the four byte-lane(s) to write during store operations. When the Nios II core is configured with a data cache line size greater than four bytes, the data master port supports pipelined Avalon-MM transfers. When the data cache line size is only four bytes, any memory pipeline latency is perceived by the data master port as wait states. Load and store

operations can complete in a single clock cycle when the data master port is connected to zero-wait-state memory.

The Nios II architecture supports on-chip cache memory for improving average data transfer performance when accessing slower memory. Refer to the "Cache Memory" section of this chapter for details.

The Nios II architecture supports tightly-coupled memory, which provides guaranteed low-latency access to on-chip memory. Refer to "Tightly-Coupled Memory" section of this chapter for details.

Related Information

- **Cache Memory** on page 15
- **Tightly-Coupled Memory** on page 16

Shared Memory for Instructions and Data

Usually the instruction and data master ports share a single memory that contains both instructions and data. While the processor core has separate instruction and data buses, the overall Nios II processor system might present a single, shared instruction/data bus to the outside world. The outside view of the Nios II processor system depends on the memory and peripherals in the system and the structure of the system interconnect fabric.

The data and instruction master ports never cause a gridlock condition in which one port starves the other. For highest performance, assign the data master port higher arbitration priority on any memory that is shared by both instruction and data master ports.

Cache Memory

The Nios II architecture supports cache memories on both the instruction master port (instruction cache) and the data master port (data cache). Cache memory resides on-chip as an integral part of the Nios II processor core. The cache memories can improve the average memory access time for Nios II processor systems that use slow off-chip memory such as SDRAM for program and data storage.

The instruction and data caches are enabled perpetually at run-time, but methods are provided for software to bypass the data cache so that peripheral accesses do not return cached data. Cache management and cache coherency are handled by software. The Nios II instruction set provides instructions for cache management.

Configurable Cache Memory Options

The cache memories are optional. The need for higher memory performance (and by association, the need for cache memory) is application dependent. Many applications require the smallest possible processor core, and can trade-off performance for size.

A Nios II processor core might include one, both, or neither of the cache memories. Furthermore, for cores that provide data and/or instruction cache, the sizes of the cache memories are user-configurable. The inclusion of cache memory does not affect the functionality of programs, but it does affect the speed at which the processor fetches instructions and reads/writes data.

Effective Use of Cache Memory

The effectiveness of cache memory to improve performance is based on the following premises:

- Regular memory is located off-chip, and access time is long compared to on-chip memory
- The largest, performance-critical instruction loop is smaller than the instruction cache
- The largest block of performance-critical data is smaller than the data cache

Optimal cache configuration is application specific, although you can make decisions that are effective across a range of applications. For example, if a Nios II processor system includes only fast, on-chip memory (i.e., it never accesses slow, off-chip memory), an instruction or data cache is unlikely to offer any performance gain. As another example, if the critical loop of a program is 2 KB, but the size of the instruction cache is 1 KB, an instruction cache does not improve execution speed. In fact, an instruction cache may degrade performance in this situation.

If an application always requires certain data or sections of code to be located in cache memory for performance reasons, the tightly-coupled memory feature might provide a more appropriate solution. Refer to the "Tightly-Coupled Memory" section for details.

Cache Bypass Methods

The Nios II architecture provides the following methods for bypassing the data cache:

- I/O load and store instructions
- Bit-31 cache bypass

I/O Load and Store Instructions Method

The load and store I/O instructions such as `ldio` and `stio` bypass the data cache and force an Avalon-MM data transfer to a specified address.

The Bit-31 Cache Bypass Method

The bit-31 cache bypass method on the data master port uses bit 31 of the address as a tag that indicates whether the processor should transfer data to/from cache, or bypass it. This is a convenience for software, which might need to cache certain addresses and bypass others. Software can pass addresses as parameters between functions, without having to specify any further information about whether the addressed data is cached or not.

To determine which cores implement which cache bypass methods, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

Related Information

Nios II Core Implementation Details

Tightly-Coupled Memory

Tightly-coupled memory provides guaranteed low-latency memory access for performance-critical applications. Compared to cache memory, tightly-coupled memory provides the following benefits:

- Performance similar to cache memory
- Software can guarantee that performance-critical code or data is located in tightly-coupled memory
- No real-time caching overhead, such as loading, invalidating, or flushing memory

Physically, a tightly-coupled memory port is a separate master port on the Nios II processor core, similar to the instruction or data master port. A Nios II core can have zero, one, or multiple tightly-coupled memories. The Nios II architecture supports tightly-coupled memory for both instruction and data access. Each tightly-coupled memory port connects directly to exactly one memory with guaranteed low, fixed latency. The memory is external to the Nios II core and is located on chip.

Accessing Tightly-Coupled Memory

Tightly-coupled memories occupy normal address space, the same as other memory devices connected via system interconnect fabric. The address ranges for tightly-coupled memories (if any) are determined at system generation time.

Software accesses tightly-coupled memory using regular load and store instructions. From the software's perspective, there is no difference accessing tightly-coupled memory compared to other memory.

Effective Use of Tightly-Coupled Memory

A system can use tightly-coupled memory to achieve maximum performance for accessing a specific section of code or data. For example, interrupt-intensive applications can place exception handler code into a tightly-coupled memory to minimize interrupt latency. Similarly, compute-intensive digital signal processing (DSP) applications can place data buffers into tightly-coupled memory for the fastest possible data access.

If the application's memory requirements are small enough to fit entirely on chip, it is possible to use tightly-coupled memory exclusively for code and data. Larger applications must selectively choose what to include in tightly-coupled memory to maximize the cost-performance trade-off.

Related Information

Using Tightly Coupled Memory with the Nios II Processor Tutorial
For additional tightly-coupled memory guidelines, refer to the *Using Tightly Coupled Memory with the Nios II Processor tutorial.*

Address Map

The address map for memories and peripherals in a Nios II processor system is design dependent. You specify the address map in Qsys.

There are three addresses that are part of the processor and deserve special mention:

- Reset address
- Exception address
- Break handler address

Programmers access memories and peripherals by using macros and drivers. Therefore, the flexible address map does not affect application developers.

Memory Management Unit

The optional Nios II MMU provides the following features and functionality:

- Virtual to physical address mapping
- Memory protection
- 32-bit virtual and physical addresses, mapping a 4-GB virtual address space into as much as 4 GB of physical memory
- 4-KB page and frame size
- Low 512 MB of physical address space available for direct access
- Hardware translation lookaside buffers (TLBs), accelerating address translation
  - Separate TLBs for instruction and data accesses
  - Read, write, and execute permissions controlled per page
  - Default caching behavior controlled per page
  - TLBs acting as n-way set-associative caches for software page tables
  - TLB sizes and associativities configurable in the Nios II Processor parameter editor
- Format of page tables (or equivalent data structures) determined by system software
- Replacement policy for TLB entries determined by system software
- Write policy for TLB entries determined by system software

For more information about the MMU implementation, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

You can optionally include the MMU when you instantiate the Nios II processor in your Nios II hardware system. When present, the MMU is always enabled, and the data and instruction caches are virtually-indexed, physically-tagged caches. Several parameters are available, allowing you to optimize the MMU for your system needs.

For complete details about user-selectable parameters for the Nios II MMU, refer to the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook*.

Note: The Nios II MMU is optional and mutually exclusive from the Nios II MPU. Nios II systems can include either an MMU or MPU, but cannot include both an MMU and MPU on the same Nios II processor core.

Related Information

- **Programming Model**
- **Instantiating the Nios II Processor**

Memory Protection Unit

The optional Nios II MPU provides the following features and functionality:

- Memory protection
- Up to 32 instruction regions and 32 data regions
- Variable instruction and data region sizes
- Amount of region memory defined by size or upper address limit
- Read and write access permissions for data regions
- Execute access permissions for instruction regions
- Overlapping regions

For more information about the MPU implementation, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

You can optionally include the MPU when you instantiate the Nios II processor in your Nios II hardware system. When present, the MPU is always enabled. Several parameters are available, allowing you to optimize the MPU for your system needs.

For complete details about user-selectable parameters for the Nios II MPU, refer to the *Instantiating the Nios II Processor* chapter of the *Nios II Processor Reference Handbook*.

Note: The Nios II MPU is optional and mutually exclusive from the Nios II MMU. Nios II systems can include either an MPU or MMU, but cannot include both an MPU and MMU on the same Nios II processor core.

Related Information

- **Programming Model**
- **Instantiating the Nios II Processor**

JTAG Debug Module

The Nios II architecture supports a JTAG debug module that provides on-chip emulation features to control the processor remotely from a host PC. PC-based software debugging tools communicate with the JTAG debug module and provide facilities, such as the following features:

- Downloading programs to memory
- Starting and stopping execution
- Setting breakpoints and watchpoints
- Analyzing registers and memory
- Collecting real-time execution trace data

Note: The Nios II MMU does not support the JTAG debug module trace.

The debug module connects to the JTAG circuitry in an Altera FPGA. External debugging probes can then access the processor via the standard JTAG interface on the FPGA. On the processor side, the debug module connects to signals inside the processor core. The debug module has nonmaskable control over the processor, and does not require a software stub linked into the application under test. All system resources visible to the processor in supervisor mode are available to the debug module. For trace data collection, the debug module stores trace data in memory either on-chip or in the debug probe.

The debug module gains control of the processor either by asserting a hardware break signal, or by writing a break instruction into program memory to be executed. In both cases, the processor transfers execution to the routine located at the break address. The break address is specified with the Nios II Processor parameter editor in Qsys.

Soft processor cores such as the Nios II processor offer unique debug capabilities beyond the features of traditional, fixed processors. The soft nature of the Nios II processor allows you to debug a system in development using a full-featured debug core, and later remove the debug features to conserve logic resources. For the release version of a product, the JTAG debug module functionality can be reduced, or removed altogether.

The following sections describe the capabilities of the Nios II JTAG debug module hardware. The usage of all hardware features is dependent on host software, such as the Nios II Software Build Tools for Eclipse, which manages the connection to the target processor and controls the debug process.

JTAG Target Connection

The JTAG target connection provides the ability to connect to the processor through the standard JTAG pins on the Altera FPGA. This provides basic capabilities to start and stop the processor, and examine and edit registers and memory. The JTAG target connection is the minimum requirement for the Nios II flash programmer.

Note: While the processor has no minimum clock frequency requirements, Altera recommends that your design's system clock frequency be at least four times the JTAG clock frequency to ensure that the on-chip instrumentation (OCI) core functions properly.

Download and Execute Software

Downloading software refers to the ability to download executable code and data to the processor's memory via the JTAG connection. After downloading software to memory, the JTAG debug module can then exit debug mode and transfer execution to the start of executable code.

Software Breakpoints

Software breakpoints allow you to set a breakpoint on instructions residing in RAM. The software breakpoint mechanism writes a break instruction into executable code stored in RAM. When the processor executes the break instruction, control is transferred to the JTAG debug module.

Hardware Breakpoints

Hardware breakpoints allow you to set a breakpoint on instructions residing in nonvolatile memory, such as flash memory. The hardware breakpoint mechanism continuously monitors the processor's current instruction address. If the instruction address matches the hardware breakpoint address, the JTAG debug module takes control of the processor.

Hardware breakpoints are implemented using the JTAG debug module's hardware trigger feature.

Hardware Triggers

Hardware triggers activate a debug action based on conditions on the instruction or data bus during real-time program execution. Triggers can do more than halt processor execution. For example, a trigger can be used to enable trace data collection during real-time processor execution.

Hardware trigger conditions are based on either the instruction or data bus. Trigger conditions on the same bus can be logically ANDed, enabling the JTAG debug module to trigger, for example, only on write cycles to a specific address.

Table 5: Trigger Conditions

Condition	Bus	Description
Specific address	Data, Instruction	Trigger when the bus accesses a specific address.
Specific data value	Data	Trigger when a specific data value appears on the bus.
Read cycle	Data	Trigger on a read bus cycle.
Write cycle	Data	Trigger on a write bus cycle.
Armed	Data, Instruction	Trigger only after an armed trigger event. Refer to the Armed Triggers section.
Range	Data	Trigger on a range of address values, data values, or both. Refer to the Triggering on Ranges of Values section.

When a trigger condition occurs during processor execution, the JTAG debug module triggers an action, such as halting execution, or starting trace capture. The table below lists the trigger actions supported by the Nios II JTAG debug module.

Table 6: Trigger Actions

Action	Description
Break	Halt execution and transfer control to the JTAG debug module.
External trigger	Assert a trigger signal output. This trigger output can be used, for example, to trigger an external logic analyzer.
Trace on	Turn on trace collection.
Trace off	Turn off trace collection.
Trace sample	Store one sample of the bus to trace buffer.
Arm	Enable an armed trigger.

Note: For the *Trace sample* triger action, only conditions on the data bus can trigger this action.

Armed Triggers

The JTAG debug module provides a two-level trigger capability, called armed triggers. Armed triggers enable the JTAG debug module to trigger on event B, only after event A. In this example, event A causes a trigger action that enables the trigger for event B.

Triggering on Ranges of Values

The JTAG debug module can trigger on ranges of data or address values on the data bus. This mechanism uses two hardware triggers together to create a trigger condition that activates on a range of values within a specified range.

Trace Capture

Trace capture refers to ability to record the instruction-by-instruction execution of the processor as it executes code in real-time. The JTAG debug module offers the following trace features:

- Capture execution trace (instruction bus cycles).
- Capture data trace (data bus cycles).
- For each data bus cycle, capture address, data, or both.
- Start and stop capturing trace in real time, based on triggers.
- Manually start and stop trace under host control.
- Optionally stop capturing trace when trace buffer is full, leaving the processor executing.
- Store trace data in on-chip memory buffer in the JTAG debug module. (This memory is accessible only through the JTAG connection.)
- Store trace data to larger buffers in an off-chip debug probe.

Certain trace features require additional licensing or debug tools from third-party debug providers. For example, an on-chip trace buffer is a standard feature of the Nios II processor, but using an off-chip trace buffer requires additional debug software and hardware provided by Imagination Technologies™, LLC or Lauterbach GmbH.

Related Information

Lauterbach.com
For more information, refer to the Lauterbach GmbH website.

Execution vs. Data Trace

The JTAG debug module supports tracing the instruction bus (execution trace), the data bus (data trace), or both simultaneously. Execution trace records only the addresses of the instructions executed, enabling you to analyze where in memory (that is, in which functions) code executed. Data trace records the data associated with each load and store operation on the data bus.

The JTAG debug module can filter the data bus trace in real time to capture the following:

- Load addresses only
- Store addresses only
- Both load and store addresses
- Load data only
- Load address and data
- Store address and data
- Address and data for both loads and stores
- Single sample of the data bus upon trigger event

Trace Frames

A frame is a unit of memory allocated for collecting trace data. However, a frame is not an absolute measure of the trace depth.

To keep pace with the processor executing in real time, execution trace is optimized to store only selected addresses, such as branches, calls, traps, and interrupts. From these addresses, host-side debug software can later reconstruct an exact instruction-by-instruction execution trace. Furthermore, execution trace data is stored in a compressed format, such that one frame represents more than one instruction. As a result of these optimizations, the actual start and stop points for trace collection during execution might vary slightly from the user-specified start and stop points.

Data trace stores 100% of requested loads and stores to the trace buffer in real time. When storing to the trace buffer, data trace frames have lower priority than execution trace frames. Therefore, while data frames are always stored in chronological order, execution and data trace are not guaranteed to be exactly synchronized with each other.

Document Revision History

Table 7: Document Revision History

Date	Version	Changes
April 2015	2015.04.02	Maintenance release.
February 2014	13.1.0	• Added information on ECC support. • Added information on enhanced floating-point custom instructions. • Removed HardCopy information. • Removed references to SOPC Builder.
May 2011	11.0.0	• Added references to new Qsys system integration tool. • Moved interrupt vector custom instruction information to the *Instantiating the Nios II Processor* chapter.
December 2010	10.1.0	Added reference to tightly-coupled memory tutorial.
July 2010	10.0.0	Maintenance release.
November 2009	9.1.0	• Added external interrupt controller interface information. • Added shadow register set information.
March 2009	9.0.0	Maintenance release.
November 2008	8.1.0	• Expanded floating-point instructions information. • Updated description of optional `cpu_resetrequest` and `cpu_resettaken` signals. • Added description of optional `debugreq` and `debugack` signals.
May 2008	8.0.0	Added MMU and MPU sections.
October 2007	7.2.0	Maintenance release.

Date	Version	Changes
May 2007	7.1.0	• Added table of contents to Introduction section. • Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	Described interrupt vector custom instruction.
May 2006	6.0.0	• Added description of optional `cpu_resetrequest` and `cpu_resettaken`. • Added section on single precision floating-point instructions.
October 2005	5.1.0	Maintenance release.
May 2005	5.0.0	Added tightly-coupled memory.
December 2004	1.2	Added new control register `ctl5`.
September 2004	1.1	Updates for Nios II 1.01 release.
May 2004	1.0	Initial release.