


Introduction


This chapter describes the Nios® II Processor MegaWizard™ interface in SOPC Builder. This chapter contains the following sections:

- “Core Nios II Page” on page 4-1
- “Caches and Memory Interfaces Page” on page 4-5
- “Advanced Features Page” on page 4-7
- “JTAG Debug Module Page” on page 4-12
- “Custom Instructions Page” on page 4-15

The Nios II Processor MegaWizard interface allows you to specify the processor features for a particular Nios II hardware system. This chapter covers only the features of the Nios II processor that you can configure with the Nios II Processor MegaWizard interface. It is not a user guide for creating complete Nios II processor systems.

 To get started using SOPC Builder to design custom Nios II systems, refer to the *Nios II Hardware Development Tutorial*. Nios II development kits also provide a number of ready-made example hardware designs that demonstrate several different configurations of the Nios II processor.

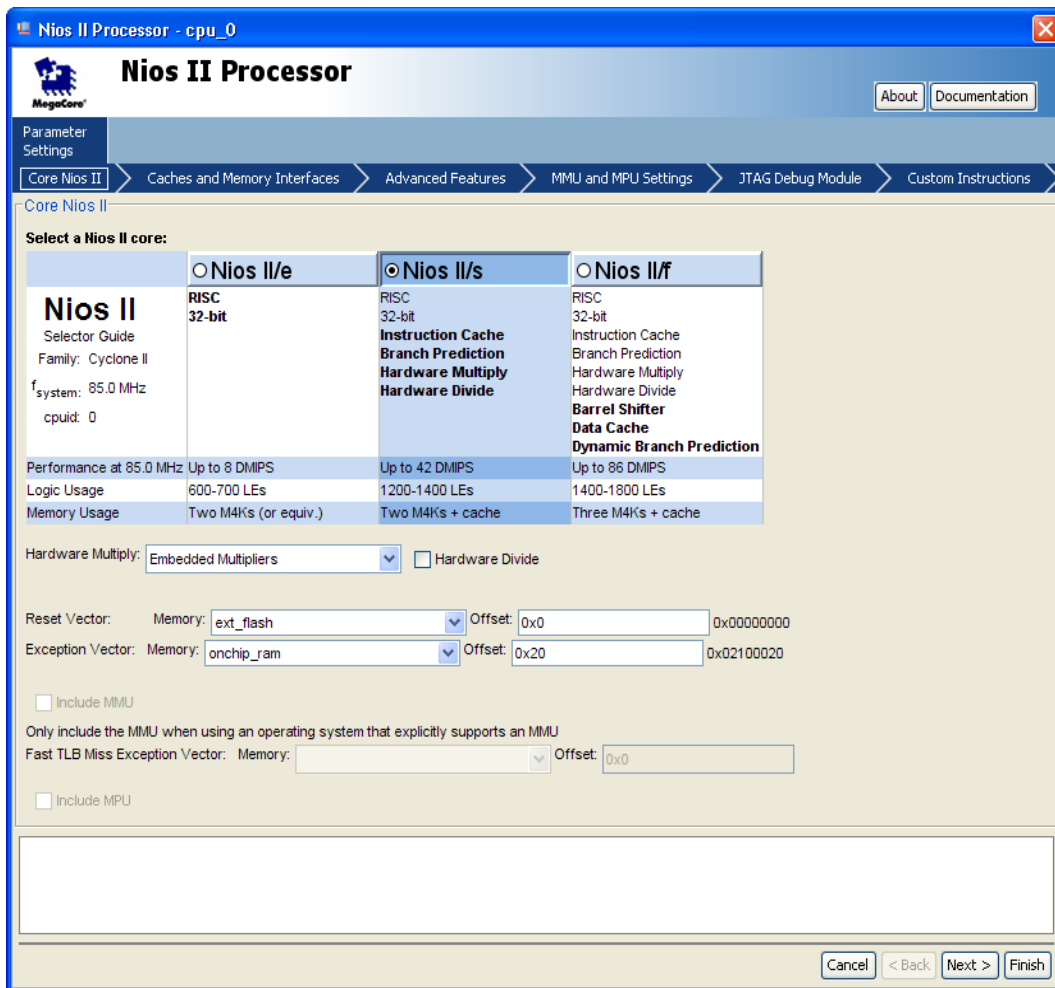
The Nios II Processor MegaWizard interface has several pages. The following sections describe the settings available on each page.

 Due to evolution and improvement of the Nios II Processor MegaWizard interface, the figures in this chapter might not match the exact screens that appear in SOPC Builder.

Core Nios II Page

The **Core Nios II** page presents the main settings for configuring the Nios II processor. [Figure 4-1](#) shows an example of the **Core Nios II** page.

Figure 4-1. Core Nios II Page in the Nios II Processor MegaWizard



The following sections describe the configuration settings available.


Core Selection

The main purpose of the **Core Nios II** page is to select the processor core. The core you select on this page affects other options available on this and other pages.

Altera offers the following Nios II cores:

- **Nios II/f**—The Nios II/f “fast” core is designed for fast performance. As a result, this core presents the most configuration options allowing you to fine-tune the processor for performance.
- **Nios II/s**—The Nios II/s “standard” core is designed for small size while maintaining performance.
- **Nios II/e**—The Nios II/e “economy” core is designed to achieve the smallest possible core size. As a result, this core has a limited feature set, and many settings are not available when the Nios II/e core is selected.

As shown in [Figure 4-1](#), the Core Nios II page displays a “selector guide” table that lists the basic properties of each core.

 For complete details of each core, refer to the [Nios II Core Implementation Details](#) chapter of the *Nios II Processor Reference Handbook*.


Multiply and Divide Settings

The Nios II/s and Nios II/f cores offer hardware multiply and divide options. You can choose the best option to balance embedded multiplier usage, logic element (LE) usage, and performance.

The **Hardware Multiply** setting for each core provides a subset of the options in the following list:

- **DSP Block**—Include DSP block multipliers in the arithmetic logic unit (ALU). This option is only present when targeting devices that have DSP block multipliers.
- **Embedded Multipliers**—Include embedded multipliers in the ALU. This option is only present when targeting FPGA devices that have embedded multipliers.
- **Logic Elements**—Include LE-based multipliers in the ALU. This option achieves high multiply performance without consuming embedded multiplier resources.
- **None**—This option conserves logic resources by eliminating multiply hardware. Multiply operations are implemented in software.

Turning on **Hardware Divide** includes LE-based divide hardware in the ALU. The **Hardware Divide** option achieves much greater performance than software emulation of divide operations.


 For details on the performance effects of the **Hardware Multiply** and **Hardware Divide** options, refer to the [Nios II Core Implementation Details](#) chapter of the *Nios II Processor Reference Handbook*.

Reset Vector

You can select the memory module where the reset code (boot loader) resides, and the location of the reset vector (reset address). The reset vector cannot be configured until your system memory components are in place.

The **Memory** list, which includes all memory modules mastered by the Nios II processor, allows you to select the reset vector memory module. In a typical system, you select a nonvolatile memory module for the reset code.

Offset allows you to specify the location of the reset vector relative to the memory module’s base address. SOPC Builder calculates the physical address of the reset vector when you modify the memory module, the offset, or the memory module’s base address, and displays the address next to the **Offset** box. This address, displayed next to the **Offset** box, is always a physical address, even when an MMU is present.

 For details on reset exceptions, refer to the [Programming Model](#) chapter of the *Nios II Processor Reference Handbook*.

General Exception Vector

You can select the memory module where the general exception vector (exception address) resides, and the location of the general exception vector. The general exception vector cannot be configured until your system memory components are in place.


The **Memory** list, which includes all memory modules mastered by the Nios II processor, allows you to select the exception vector memory module. In a typical system, you select a low-latency memory module for the exception code.

Offset allows you to specify the location of the exception vector relative to the memory module's base address. SOPC Builder calculates the physical address of the exception vector when you modify the memory module, the offset, or the memory module's base address. This address, displayed next to the **Offset** box, is always a physical address, even when an MMU is present.

For details on exceptions, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Memory Management Unit Settings

The Nios II/f core offers a memory management unit (MMU) to support full-featured operating systems. Turning on **Include MMU** includes the Nios II MMU in your Nios II hardware system.


 Do not include an MMU in your Nios II system unless your operating system requires it. The MMU is only useful with software that takes advantage of it. Many Nios II systems involve simpler system software, such as Altera® HAL or MicroC/OS-II. Such software is unlikely to function correctly with an MMU-based Nios II processor.


Fast TLB Miss Exception Vector


The fast TLB miss exception vector is a special exception vector used exclusively by the MMU to handle TLB miss exceptions. You can select the memory module where the fast TLB miss exception vector (exception address) resides, and the location of the fast TLB miss exception vector. The fast TLB miss exception vector cannot be configured until your system memory components are in place.

The **Memory** list, which includes all memory modules mastered by the Nios II processor, allows you to select the exception vector memory module. In a typical system, you select a low-latency memory module for the exception code.

Offset allows you to specify the location of the exception vector relative to the memory module's base address. SOPC Builder calculates the physical address of the exception vector when you modify the memory module, the offset, or the memory module's base address. This address, displayed next to the **Offset** box, is always a physical address.


 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 in the same design.


 For details on the Nios II MMU, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

 To function correctly with the MMU, the base physical address of all exception vectors (reset, general exception, break, and fast TLB miss) must point to low physical memory so that hardware can correctly map their virtual addresses into the kernel partition. This restriction is enforced by the Nios II Processor MegaWizard interface.

Memory Protection Unit Settings

The Nios II/f core offers a memory protection unit (MPU) to support operating systems and runtime environments that desire memory protection without the overhead of virtual memory management. Turning on **Include MPU** includes the Nios II MPU in your Nios II hardware system.

 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 in the same design.

 For details on the Nios II MPU, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Caches and Memory Interfaces Page

The **Caches and Memory Interfaces** page allows you to configure the cache and tightly-coupled memory usage for the instruction and data master ports. [Figure 4-2](#) shows an example of the **Caches and Memory Interfaces** page.

The following sections describe the configuration settings available.

Instruction Master Settings

The **Instruction Master** settings provide the following options for the Nios II/f and Nios II/s cores:

- **Instruction Cache**—Specifies the size of the instruction cache. Valid sizes are from 512 bytes to 64 KBytes, or **None**.

Choosing **None** disables the instruction cache, which also removes the Avalon-MM instruction master port from the Nios II processor. In this case, you must include a tightly-coupled instruction memory.

- **Enable Bursts**—The Nios II processor can fill its instruction cache lines using burst transfers. Usually you enable bursts on the processor's instruction master when instructions are stored in DRAM, and disable bursts when instructions are stored in SRAM.

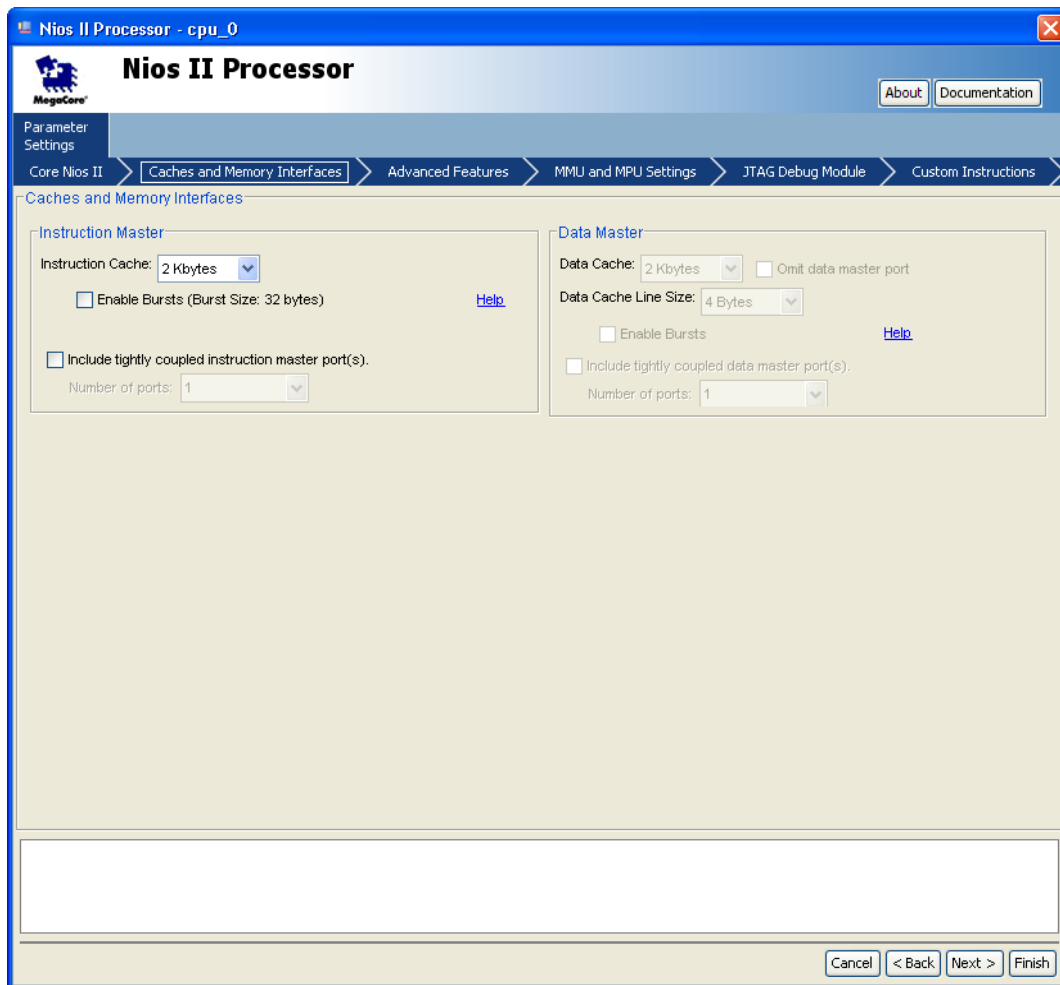
Bursting to DRAM typically improves memory bandwidth, but might consume additional FPGA resources. Be aware that when bursts are enabled, accesses to slaves might go through additional hardware (called “burst adapters”) which might decrease f_{MAX} .

When the Nios II processor transfers execution to the first word of a cache line, the processor fills the line by executing a sequence of word transfers that have ascending addresses, such as 0, 4, 8, 12, 16, 20, 24, 28.

However, when the Nios II processor transfers execution to an instruction that is not the first word of a cache line, the processor fetches the required (or “critical”) instruction first, and then fills the rest of the cache line. The addresses of a burst increase until the last word of the cache line is filled, and then continue with the first word of the cache line. For example, with a 32-byte cache line, transferring control to address 8 results in a burst with the following address sequence: 8, 12, 16, 20, 24, 28, 0, 4.

- **Include tightly coupled instruction master port(s)**—When on, the Nios II processor includes tightly-coupled memory ports. You can specify one to four ports with the **Number of ports** setting. Tightly-coupled memory ports appear on the connection panel of the Nios II processor in the SOPC Builder **System Contents** tab. You must connect each port to exactly one memory component in the system.

Figure 4–2. Caches and Memory Interfaces Page in the Nios II Processor MegaWizard



Data Master Settings

The **Data Master** settings provide the following options for the Nios II/f core:

- **Data Cache**—Specifies the size of the data cache. Valid sizes are from **512 bytes** to **64 KBytes**, or **None**. Depending on the value specified for **Data Cache**, the following options are available:
 - **Data Cache Line Size**—Valid sizes are **4 bytes**, **16 bytes**, or **32 bytes**.
 - **Omit data master port**—If you set **Data Cache** to **None**, you can optionally turn on **Omit data master port** to remove the Avalon-MM data master port from the Nios II processor. In this case, you must include a tightly-coupled data memory.



Although the Nios II processor can operate entirely out of tightly-coupled memory without the need for Avalon-MM instruction or data masters, software debug is not possible when either the Avalon-MM instruction or data master is omitted.

Enable Bursts—The Nios II processor can fill its data cache lines using burst transfers. Usually you enable bursts on the processor's data bus when processor data is stored in DRAM, and disable bursts when processor data is stored in SRAM.

Bursting to DRAM typically improves memory bandwidth but might consume additional FPGA resources. Be aware that when bursts are enabled, accesses to slaves might go through additional hardware (called “burst adapters”) which might decrease f_{MAX} .

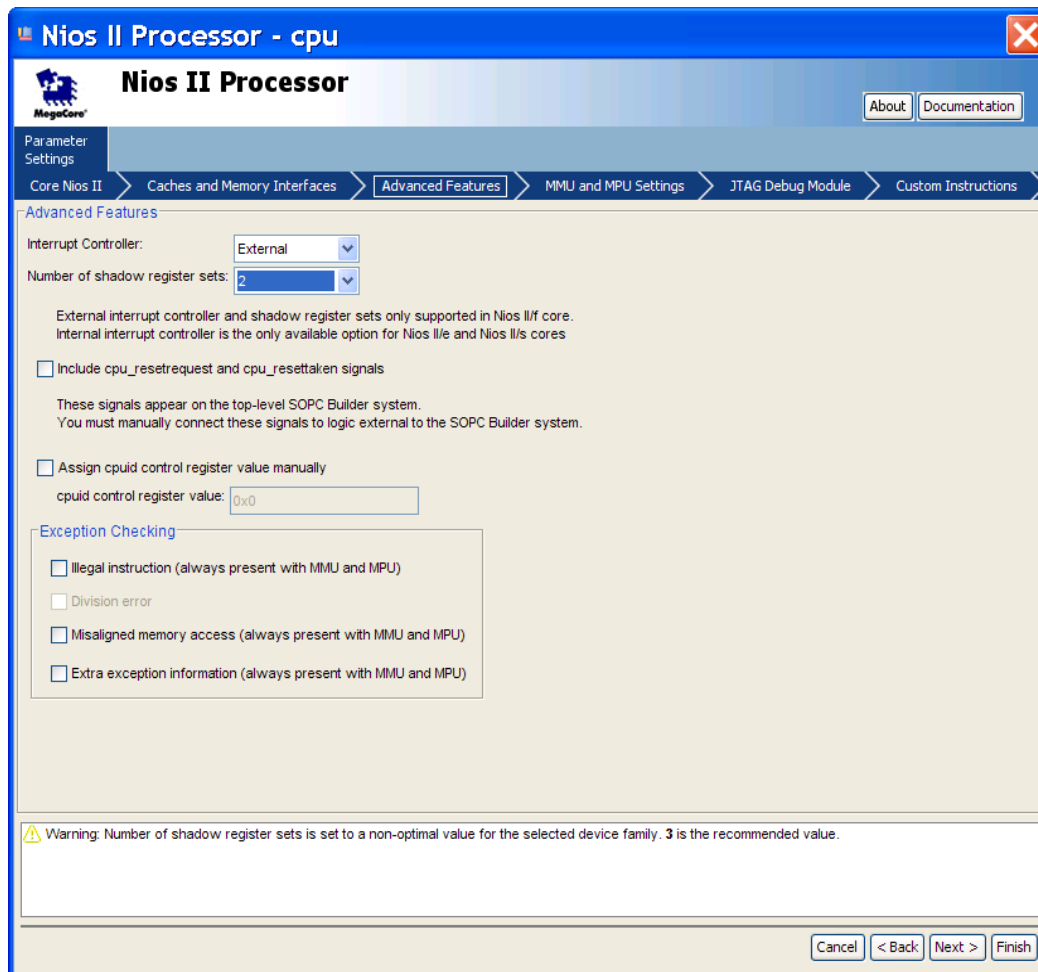
Bursting is only enabled for data line sizes greater than 4 bytes. The burst length is 4 for a 16 byte line size and 8 for a 32 byte line size. Data cache bursts are always aligned on the cache line boundary. For example, with a 32-byte Nios II data cache line, a cache miss to the address 8 results in a burst with the following address sequence: 0, 4, 8, 12, 16, 20, 24 and 28.

- **Include tightly coupled data master port(s)**—When on, the Nios II processor includes tightly-coupled memory ports. You can specify one to four ports with the **Number of ports** setting. Tightly-coupled memory ports appear on the connection panel of the Nios II processor in the SOPC Builder **System Contents** tab. You must connect each port to exactly one memory component in the system.

Advanced Features Page


The **Advanced Features** page allows you to enable specialized features of the Nios II processor. [Figure 4-3](#) shows the **Advanced Features** page.

Figure 4-3. Advanced Features Page in the Nios II Processor MegaWizard



Reset Signals

The **Include `cpu_resetrequest` and `cpu_resettaken` signals** reset signals setting provides the following functionality. When on, the Nios II processor includes processor-only reset request signals. These signals let another device individually reset the Nios II processor without resetting the entire SOPC Builder system. The signals are exported to the top level of your SOPC Builder system.

 For further details on the reset signals, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Control Registers

The **Assign `cpuid` control register value manually** control register setting provides the following functionality. When on, you can assign the `cpuid` control register value yourself. Normally, `cpuid` is automatically assigned in your SOPC Builder system. To assign the value yourself, type a 32-bit value (in hexadecimal or decimal format) into the **`cpuid` control register value** box.

Exception Checking

The **Exception Checking** settings provide the following options:

- **Illegal instruction**—When **Illegal instruction** is on, the processor generates an illegal instruction exception when an instruction with an undefined opcode or opcode-extension field is executed.



When your system contains an MMU or MPU, the processor automatically generates illegal instruction exceptions. Therefore, the **Illegal instruction** setting is always disabled when the **Core Nios II** page **Include MMU** or **Include MPU** are on.

- **Division error**—Division error detection is only available for the Nios II/f core, and only then when **Hardware Divide** on the **Core Nios II** page is on. When divide instructions are not supported by hardware, the **Division error** setting is disabled.

When **Division error** is on, the processor generates a division error exception when it detects divide instructions that produce a result that cannot be represented in the destination register. This only happens in the following two cases:


- Divide by zero
- Divide overflow—A signed division that divides the largest negative number -2147483648 (0x80000000) by -1 (0xffffffff).
- **Misaligned memory access**—Misaligned memory access detection is only available for the Nios II/f core. When **Misaligned memory access** is on, the processor checks for misaligned memory accesses.




When your system contains an MMU or MPU, the processor automatically generates misaligned memory access exceptions. Therefore, the **Misaligned memory access** checkbox is always disabled when **Include MMU** or **Include MPU** on the **Core Nios II** page are on.

There are two misaligned memory address exceptions:

- **Misaligned data address**—Data addresses of load and store instructions are checked for misalignment. A data address is considered misaligned if the byte address is not a multiple of the data width of the load or store instruction (4 bytes for word, 2 bytes for half-word). Byte load and store instructions are always aligned so never generate a misaligned data address exception.
- **Misaligned destination address**—Destination instruction addresses of `br`, `callr`, `jmp`, `ret`, `eret`, and `bret` instructions are checked for misalignment. A destination instruction address is considered misaligned if the target byte address of the instruction is not a multiple of four.
- **Extra exception information**—When **Extra exception information** is on, nonbreak exceptions store a code in the `CAUSE` field of the `exception` control register to indicate the cause of the exception.

 When your system contains an MMU or MPU, the processor automatically generates extra exception information. Therefore, the **Extra exception information** setting is always disabled when the **Core Nios II** page **Include MMU** or **Include MPU** are on.

Your exception handler can use this code to quickly determine the proper action to take, rather than have to determine the cause of an exception through instruction decoding. Additionally, some exceptions also store the instruction or data address associated with the exception in the `badaddr` register.


 For further descriptions of exceptions, exception handling, and control registers, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

External Interrupt Controller Interface

The **Interrupt controller** setting determines which of the following configurations is implemented:

- Internal interrupt controller
- External interrupt controller (EIC) interface

The EIC interface is available only on the Nios II/f core.


 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 Embedded Design Suite (EDS) version 9.0 or higher. Earlier versions have an implementation of the `eret` instruction that is incompatible with shadow register sets.


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

Shadow Register Sets

The **Number of shadow register sets** setting determines whether the Nios II core implements shadow register sets. The Nios II core can be configured with up to 63 shadow register sets.

Shadow register sets are available only on the Nios II/f core.

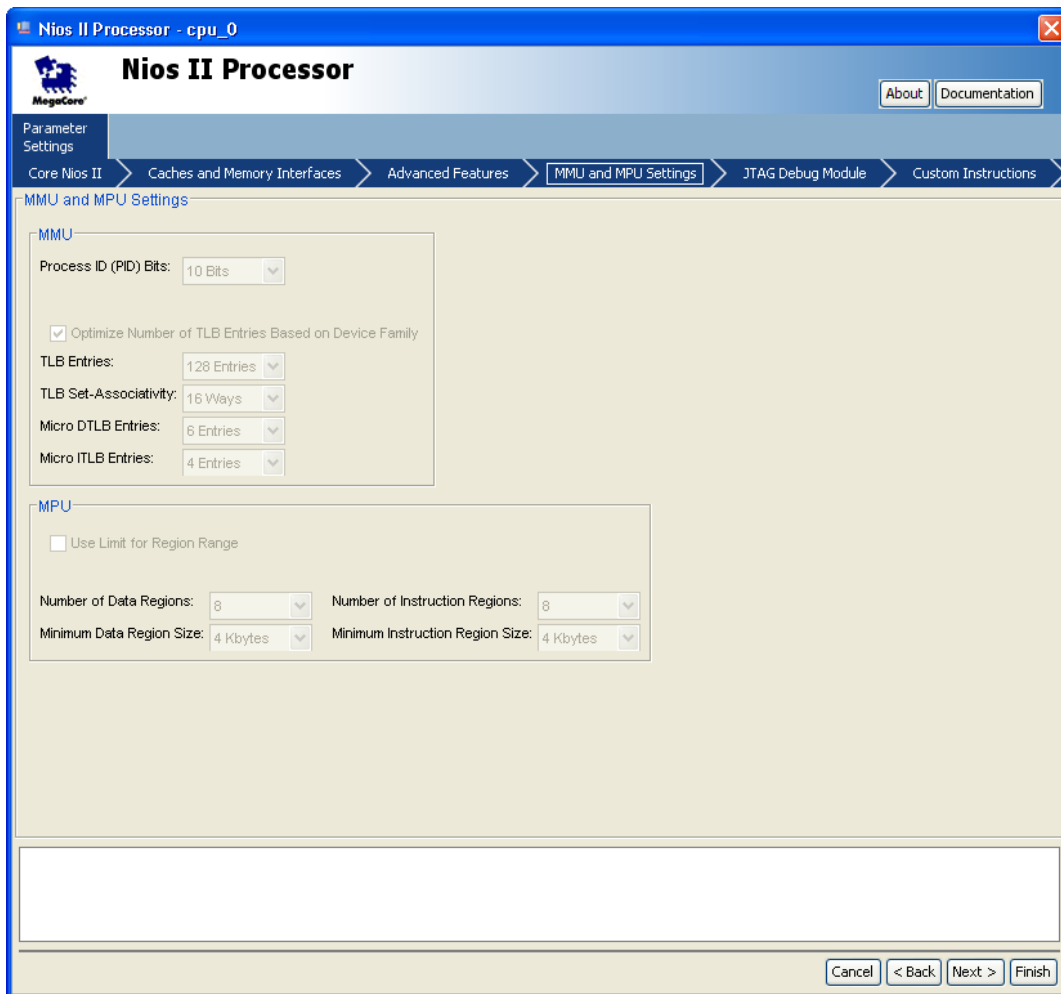
 When the EIC interface and shadow register sets are implemented on the Nios II core, you must ensure that your software is built with the Nios II EDS version 9.0 or higher.

 For details about shadow register sets, refer to “Registers” in the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

MMU and MPU Settings Page

The **MMU and MPU Settings** page presents settings for configuring the MMU and MPU on the Nios II processor. You can select the features appropriate for your target application. [Figure 4-4](#) shows the **MMU and MPU Settings** page.


Figure 4-4. MMU and MPU Settings Page in the Nios II Processor MegaWizard



MMU

When **Include MMU** on the **Core Nios II** page is on, the **MMU and MPU Settings** page provide the following options for the MMU in the Nios II/f core. Typically, you should not need to change any of these settings from their default values.


- **Process ID (PID) Bits**—Specifies the number of bits to use to represent the process identifier.
- **Optimize number of TLB entries based on device family**—When on, specifies the optimal number of TLB entries to allocate based on the device family of the target hardware and disables **TLB Entries**.
- **TLB Entries**—Specifies the number of entries in the translation lookaside buffer (TLB).
- **TLB Set-Associativity**—Specifies the number of set-associativity ways in the TLB.
- **Micro ITLB Entries**—Specifies the number of entries in the micro instruction TLB.
- **Micro DTLB Entries**—Specifies the number of entries in the micro data TLB.


 For details on the MMU, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*. For specifics on the Nios II/f core, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

MPU

When **Include MPU** on the **Core Nios II** page is on, the **MPU settings** on the **MMU and MPU Settings** page provide the following options for the MPU in the Nios II/f core.

- **Use Limit for Region Range**—Controls whether the amount of memory in the region is defined by size or by upper address limit. When on, the amount of memory is based on the given upper address limit. When off, the amount of memory is based on the given size.
- **Number of Data Regions**—Specifies the number of data regions to allocate. Allowed values range from 2 to 32.
- **Minimum Data Region Size**—Specifies the minimum data region size. Allowed values range from 64 bytes to 1 Mbyte and must be a power of two.
- **Number of Instruction Regions**—Specifies the number of instruction regions to allocate. Allowed values range from 2 to 32.
- **Minimum Instruction Region Size**—Specifies the minimum instruction region size. Allowed values range from 64 bytes to 1 Mbyte and must be a power of two.

 The maximum region size is the size of the Nios II instruction and data addresses automatically determined when the Nios II system is generated in SOPC Builder. Maximum region size is based on the address range of slaves connected to the Nios II instruction and data masters.

 For details on the MPU, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*. For specifics on the Nios II/f core, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

JTAG Debug Module Page

The **JTAG Debug Module** page presents settings for configuring the JTAG debug module on the Nios II processor. You can select the debug features appropriate for your target application.

Soft-core processors such as the Nios II processor offer unique debug capabilities beyond the features of traditional fixed processors. The soft-core 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, you might choose to reduce the JTAG debug module functionality, or remove it altogether.

Table 4-1 describes the debug features available to you for debugging your system.

Table 4-1. Debug Configuration Features


Feature	Description
JTAG Target Connection	Connects to the processor through the standard JTAG pins on the Altera FPGA. This provides the basic capabilities to start and stop the processor, and examine/edit registers and memory.
Download Software	Downloads executable code to the processor's memory via the JTAG connection.
Software Breakpoints	Sets a breakpoint on instructions residing in RAM.
Hardware Breakpoints	Sets a breakpoint on instructions residing in nonvolatile memory, such as flash memory.
Data Triggers	Triggers based on address value, data value, or read or write cycle. You can use a trigger to halt the processor on specific events or conditions, or to activate other events, such as starting execution trace, or sending a trigger signal to an external logic analyzer. Two data triggers can be combined to form a trigger that activates on a range of data or addresses.
Instruction Trace	Captures the sequence of instructions executing on the processor in real time.
Data Trace	Captures the addresses and data associated with read and write operations executed by the processor in real time.
On-Chip Trace	Stores trace data in on-chip memory.
Off-Chip Trace	Stores trace data in an external debug probe. Off-chip trace instantiates a PLL inside the Nios II core. Off-chip trace requires a debug probe from First Silicon Solutions (FS2) or Lauterbach GmbH.

The following sections describe the configuration settings available.

Debug Level Settings

There are five debug levels in the **JTAG Debug Module** page, as shown in [Figure 4-5](#).

[Table 4-2 on page 4-15](#) is a detailed list of the characteristics of each debug level. Different levels consume different amounts of on-chip resources. Certain Nios II cores have restricted debug options, and certain options require debug tools provided by First Silicon Solutions (FS2) or Lauterbach GmbH.

 For details on the debug features available, refer to the FS2 website (www.fs2.com) and the Lauterbach GmbH website (www.lauterbach.com).

Debug Signals

The **Include debugreq and debugack signals** debug signals setting provides the following functionality. When on, the Nios II processor includes debug request and acknowledge signals. These signals let another device temporarily suspend the Nios II processor for debug purposes. The signals are exported to the top level of your SOPC Builder system.


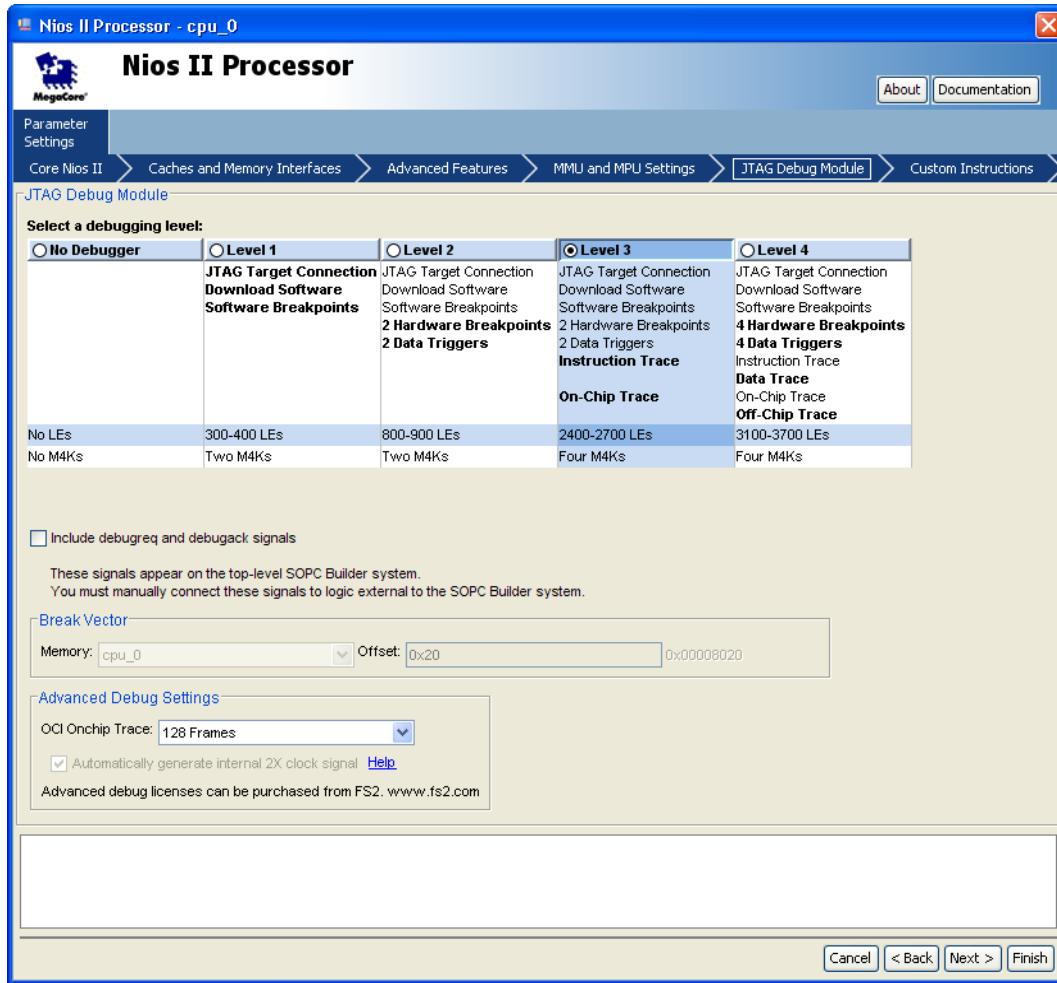
 For further details on the debug signals, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Figure 4-5. JTAG Debug Module Page in the Nios II Processor MegaWizard



Break Vector

If the Nios II processor contains a JTAG debug module, SOPC Builder determines a break vector (break address). **Memory** is always the processor core you are configuring. **Offset** is fixed at 0x20. SOPC Builder calculates the physical address of the break vector from the memory module's base address and the offset.

Advanced Debug Settings

Debug levels 3 and 4 support trace data collection into an on-chip memory buffer. You can set the on-chip trace buffer size to sizes from 128 to 64K trace frames, using **OCI Onchip Trace**. Larger buffer sizes consume more on-chip M4K RAM blocks. Every M4K RAM block can store up to 128 trace frames.



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

Table 4-2. JTAG Debug Module Levels

Debug Feature	No Debug	Level 1	Level 2	Level 3	Level 4 (1)
Logic Usage	0	300—400 LEs	800—900 LEs	2,400—2,700 LEs	3,100—3,700 LEs
On-Chip Memory Usage	0	Two M4Ks	Two M4Ks	Four M4Ks	Four M4Ks
External I/O Pins Required (2)	0	0	0	0	20
JTAG Target Connection	No	Yes	Yes	Yes	Yes
Download Software	No	Yes	Yes	Yes	Yes
Software Breakpoints	None	Unlimited	Unlimited	Unlimited	Unlimited
Hardware Execution Breakpoints	0	None	2	2	4
Data Triggers	0	None	2	2	4
On-Chip Trace	0	None	None	Up to 64K Frames (3)	Up to 64K Frames
Off-Chip Trace (4)	0	None	None	None	128K Frames

Notes to Table 4-2:

- (1) Level 4 requires the purchase of a software upgrade from FS2 or Lauterbach.
- (2) Not including the dedicated JTAG pins on the Altera FPGA.
- (3) An additional license from FS2 is required to use more than 16 frames.
- (4) Off-chip trace requires the purchase of additional hardware from FS2 or Lauterbach.

Debug level 4 also supports manual 2X clock signal specification. If you want to use a specific 2X clock signal in your FPGA design, turn off **Automatically generate internal 2X clock signal** and drive a 2X clock signal into your SOPC Builder system manually.



For further details on trace frames, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Custom Instructions Page

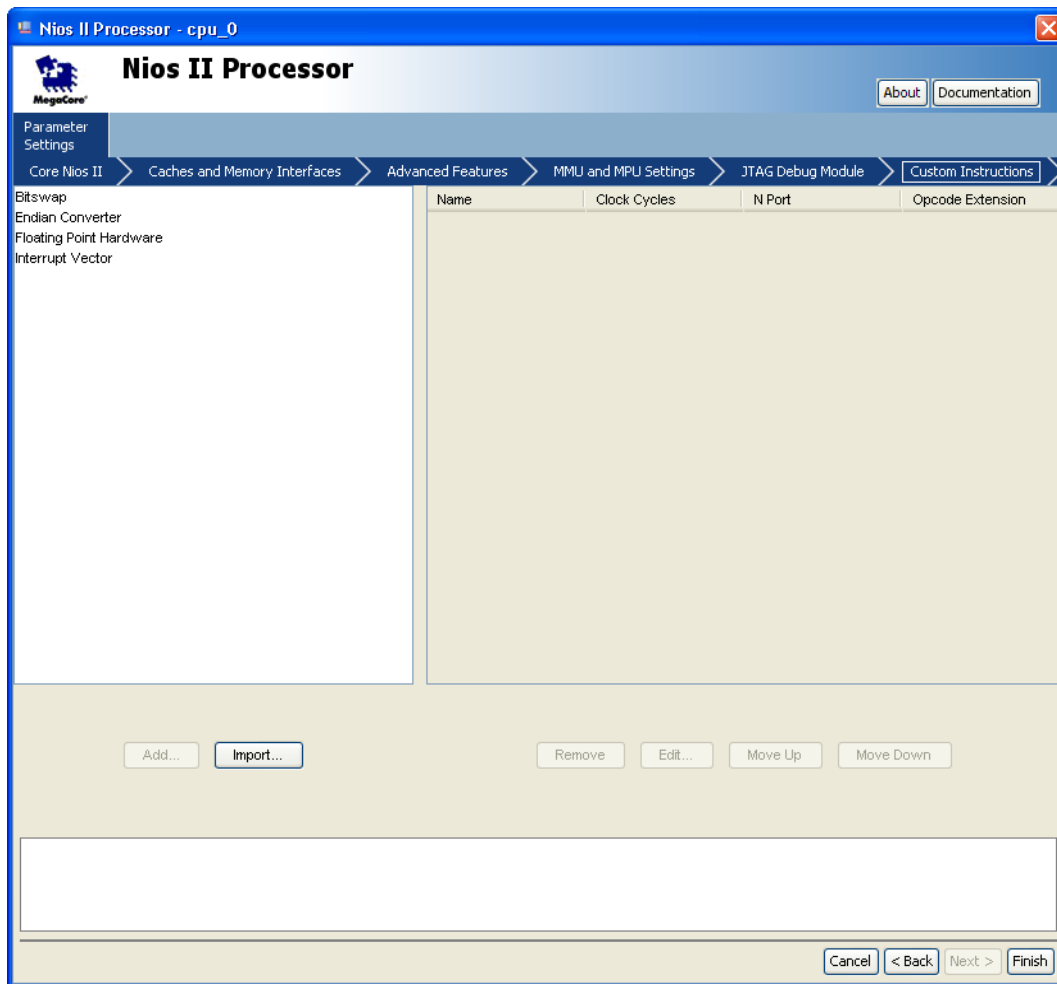
The **Custom Instructions** page allows you to connect custom instruction logic to the Nios II arithmetic logic unit (ALU). You can achieve significant performance improvements, often on the order of 10x to 100x, by implementing performance-critical operations in hardware using custom instruction logic. [Figure 4-6](#) shows an example of the **Custom Instructions** page.


To add a custom instruction to the Nios II processor, select the custom instruction from the list at the left side of the page, and click **Add**. The added instruction appears on the right side of the page.




To display custom instructions in the table of active components on the SOPC Builder **System Contents** tab, click **Filter** in the lower right of the **System Contents** tab, and turn on **Nios Custom Instruction**.

To create your own custom instruction using the component editor, click **Import**. After finishing in the component editor, the new instruction appears in the list at the left side of the **Custom Instructions** page.

Figure 4-6. Custom Instructions Page in the Nios II Processor MegaWizard

 All signals in Nios II custom instructions must have the **Custom Instruction Slave** interface type. To guarantee the component editor automatically selects the **Custom Instruction Slave** interface type for your signals correctly during import, begin your signal names with the prefix `ncs_`. This prefix allows the component editor to determine the connection point type: a Nios II custom instruction slave. For example, if a custom instruction component has two data signals plus clock, reset, and result signals, an appropriate set of signal names is `ncs_dataaa`, `ncs_datadb`, `ncs_clk`, `ncs_reset`, and `ncs_result`.

 A complete discussion of the hardware and software design process for custom instructions is beyond the scope of this chapter. For full details on the topic of custom instructions, including working example designs, refer to the *Nios II Custom Instruction User Guide*.

The following sections describe the default custom instructions Altera provides.

Interrupt Vector Custom Instruction

The Nios II processor offers an interrupt vector custom instruction which reduces average and worst case interrupt latency.

To add the interrupt vector custom instruction to the Nios II processor, select **Interrupt Vector** from the list, and click **Add**.

There can only be one interrupt vector custom instruction component in a Nios II processor. If the interrupt vector custom instruction is present in the Nios II processor, the hardware abstraction layer (HAL) source detects it at compile time and generates code using the custom instruction.

The interrupt vector custom instruction improves both average and worst case interrupt latency by up to 20%. To achieve the lowest possible interrupt latency, consider using tightly-coupled memories so that interrupt handlers can run without cache misses.



The interrupt vector custom instruction is not compatible with the EIC interface. For the Nios II/f core, the EIC interface with the Altera vectored interrupt controller component provides superior performance.



For details of the interrupt vector custom instruction implementation, refer to “Exception and Interrupt Controller” in the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*. For guidance with tightly-coupled memories, refer to “Tightly-Coupled Memory” in the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Floating-Point Hardware Custom Instruction

The Nios II processor offers a set of optional predefined custom instructions that implement floating-point arithmetic operations. You can include these custom instructions to support computation-intensive floating-point applications.

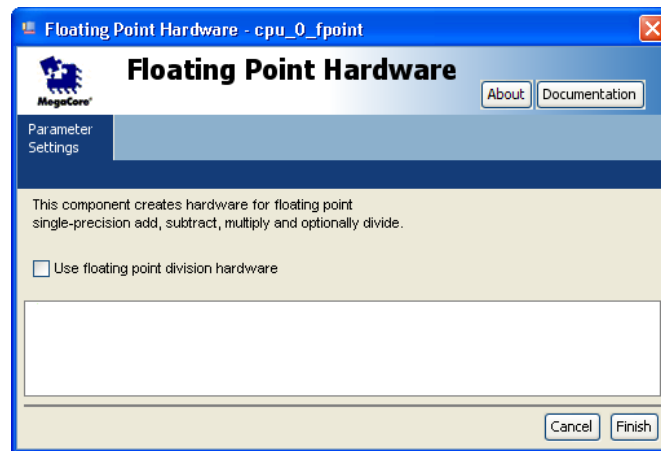
The basic set of floating-point custom instructions includes single precision (32-bit) floating-point addition, subtraction, and multiplication. Floating-point division is available as an extension to the basic instruction set. The best choice for your hardware design depends on a balance among floating-point usage, hardware resource usage, and performance.

If the target device includes on-chip multiplier blocks, the floating-point custom instructions incorporate them as needed. If there are no on-chip multiplier blocks, the floating-point custom instructions are entirely based on general-purpose logic elements.




The opcode extensions for the floating-point custom instructions are 252 through 255 (0xFC through 0xFF). These opcode extensions cannot be modified.

To add the floating-point custom instructions to the Nios II processor, select **Floating Point Hardware** from the list, and click **Add**. By default, SOPC Builder includes floating-point addition, subtraction, and multiplication, but omits the more resource intensive floating-point division. The **Floating Point Hardware** wizard, shown in [Figure 4-7](#), appears, giving you the option to include the floating-point division hardware.

Figure 4-7. Floating Point Hardware Wizard

Turn on **Use floating point division hardware** to include floating-point division hardware. The floating-point division hardware requires more resources than the other instructions, so you might wish to omit it if your application does not make heavy use of floating-point division.

Click **Finish** to add the floating-point custom instructions to the Nios II processor.

 For further details on the floating-point custom instructions, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Endian Converter Custom Instruction

The Nios II processor core offers an endian converter custom instruction to reduce the time spent performing byte reversal operations.

To add the endian converter custom instruction to the Nios II processor, select **Endian Converter** from the list, and click **Add**.

The endian converter custom instruction takes a 32 bit value and converts the endianness in a single clock cycle. The Nios II processor core supports little endian so this custom instruction allows you to convert data shared with a big endian processor core. It is important to note that this custom instruction does not convert the Nios II processor core to big endian architecture, it only converts big endian data to little endian and vice versa.

Bitswap Custom Instruction

The Nios II processor core offers a bitswap custom instruction to reduce the time spent performing bit reversal operations.

To add the bitswap custom instruction to the Nios II processor, select **Bitswap** from the list, and click **Add**.

The bitswap custom instruction reverses a 32 bit value in a single clock cycle. To perform the equivalent operation in software requires many mask and shift operations.

 For details about integrating the bitswap custom instruction into your own algorithm, refer to the *Nios II Custom Instruction User Guide*.

The Quartus II IP File

The Quartus® II IP file (.qip) is a file generated by the MegaWizard interface or SOPC Builder that contains information about a generated IP core. You are prompted to add this .qip file to the current project at the time of Quartus II file generation. In most cases, the .qip file contains all of the necessary assignments and information required to process the core or system in the Quartus II compiler. Generally, a single .qip file is generated for each MegaCore function and for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate .qip file, so the system .qip file references the component .qip file.

Referenced Documents

This chapter references the following documents:

- *Nios II Hardware Development Tutorial*
- *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*
- *Programming Model* chapter of the *Nios II Processor Reference Handbook*
- *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*
- *Nios II Custom Instruction User Guide*

Document Revision History

Table 4-3 shows the revision history for this document.

Table 4-3. Document Revision History (Part 1 of 2)

Date & Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	<ul style="list-style-type: none"> ■ Added external interrupt controller interface information. ■ Added shadow register set information. 	Added shadow register sets and external interrupt controller support
March 2009 v9.0.0	Maintenance release.	—
November 2008 v8.1.0	<ul style="list-style-type: none"> ■ Added debugreq and debugack signal options to Advanced Features page. ■ Added cpuid manual override options to Advanced Features page. 	—
May 2008 v8.0.0	<ul style="list-style-type: none"> ■ Added MMU options to Nios II Core and Advanced Features pages. ■ Added exception handling options Advanced Features page. 	Added MMU and exception handling options.
October 2007 v7.2.0	Changed title to match other Altera documentation.	—

Table 4-3. Document Revision History (Part 2 of 2)

Date & Document Version	Changes Made	Summary of Changes
May 2007 v7.1.0	<ul style="list-style-type: none"> ■ Revised to reflect new MegaWizard interface. ■ Added “Endian Converter Custom Instruction” on page 4–18 and “Bitswap Custom Instruction” on page 4–18. ■ Added table of contents to Introduction section. ■ Added Referenced Documents section. 	—
March 2007 v7.0.0	Maintenance release.	—
November 2006 v6.1.0	<ul style="list-style-type: none"> ■ Add section on interrupt vector custom instruction. ■ Add section on system-dependent Nios II processor settings. 	—
May 2006 v6.0.0	<ul style="list-style-type: none"> ■ Added details on floating-point custom instructions. ■ Added section on Advanced Features tab. 	—
October 2005 v5.1.0	Maintenance release.	—
May 2005 v5.0.0	<ul style="list-style-type: none"> ■ Updates to reflect new GUI options in Nios II processor version 5.0. ■ New details in “Caches and Tightly-Coupled Memory” section. 	—
September 2004 v1.1	<ul style="list-style-type: none"> ■ Updates to reflect new GUI options in Nios II processor version 1.1. ■ New details in section “Multiply and Divide Settings.” 	—
May 2004 v1.0	Initial release.	—