

Instantiating the Nios II Processor

2015.04.02

NIIS1004



Subscribe



Send Feedback

This chapter describes the Nios[®] II Processor parameter editor in Qsys. The Nios II Processor parameter editor allows you to specify the processor features for a particular Nios II hardware system. This chapter covers the features of the Nios II processor that you can configure with the Nios II Processor parameter editor; it is not a user guide for creating complete Nios II processor systems.

To get started designing custom Nios II systems, refer to the Nios II Hardware Development Tutorial.

Development kits for Altera devices, available on the All Development Kits page of the Altera website, also provide ready-made hardware design examples that demonstrate different configurations of the Nios II processor.

Related Information

- [All Development Kits](#)
- [Nios II Hardware Development Tutorial](#)

To get started designing custom Nios II systems, refer to the Nios II Hardware Development Tutorial.

Core Nios II Tab

The **Core Nios II** tab presents the main settings for configuring the Nios II processor.

Table 1: Core Nios II Tab Parameters

Name	Description
Select a Nios II Core	
Nios II Core	Refer to the "Core Selection" section.
Hardware Arithmetic Operation	
Hardware multiplication type	Refer to the "Multiply and Divide Settings" section.
Hardware divide	
Reset Vector	
Reset vector memory	Refer to the "Reset Vector" section.
Reset vector offset	
Reset vector	
Exception Vector	

© 2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO
9001:2008
Registered



Name	Description
Exception vector memory	Refer to the "General Exception Vectors" section.
Exception vector offset	
Exception vector	
MMU and MPU	
Include MMU	Refer to the "Memory Management Unit Settings" section.
Fast TLB Miss Exception vector memory	
Fast TLB Miss Exception vector offset	
Fast TLB Miss Exception vector	
Include MPU	Refer to the "Memory Protection Unit Settings" section.

The following sections describe the configuration settings available.

Related Information

- [Memory Management Unit Settings](#) on page 4
- [General Exception Vector](#) on page 3
- [Multiply and Divide Settings](#) on page 2
- [Reset Vector](#) on page 3
- [Core Selection](#) on page 2
- [Memory Protection Unit Settings](#) on page 5

Core Selection

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

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.

The **Core Nios II** tab displays a selector guide table that lists the basic properties of each core.

For implementation information about each core, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Nios II Core Implementation Details](#)

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 multiplication type** parameter for each core provides the following list:

- **DSP Block**—Include DSP block multipliers in the arithmetic logic unit (ALU). This option is only selectable 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, but with reduced f_{MAX} .
- **None**—This option conserves logic resources by eliminating multiply hardware. Multiply operations are implemented in software.

Note: Shift operations use the multiplier. So, **Hardware multiplication type** affects shift instruction speed.

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 information about the performance effects of the hardware multiply and divide options, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Nios II Core Implementation Details](#)

Reset Vector

Parameters in this section 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 **Reset vector memory** list, which includes all memory modules mastered by the Nios II processor, selects the reset vector memory module. In a typical system, select a nonvolatile memory module for the reset code.

Note: Qsys provides an **Absolute** option, which allows you to specify an absolute address in **Reset vector offset**. Use an absolute address when the memory storing the reset handler is located outside of the processor system and subsystems of the processor system.

Reset vector offset specifies the location of the reset vector relative to the memory module's base address. Qsys calculates the physical address of the reset vector when you modify the memory module, the offset, or the memory module's base address. In Qsys, **Reset vector** displays the read-only, calculated address. The address is always a physical address, even when an MMU is present.

For information about reset exceptions, refer to the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Programming Model](#)

General Exception Vector

Parameters in this section 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 **Exception vector memory** list, which includes all memory modules mastered by the Nios II processor, selects the exception vector memory module. In a typical system, select a low-latency memory module for the exception code.

Note: Qsys provides an **Absolute** option, which allows you to specify an absolute address in **Exception vector offset**. Use an absolute address when the memory storing the exception handler is located outside of the processor system and subsystems of the processor system.

Exception vector offset specifies the location of the exception vector relative to the memory module's base address. Qsys calculates the physical address of the exception vector when you modify the memory module, the offset, or the memory module's base address. In Qsys, **Exception vector** displays the read-only, calculated address. The address is always a physical address, even when an MMU is present.

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

Related Information

[Programming Model](#)

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.

Note: 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. Parameters in this section 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 Fast TLB Miss Exception vector memory list, which includes all memory modules mastered by the Nios II processor, selects the exception vector memory module. In a typical system, select a low-latency memory module for the exception code.

Note: Qsys provides an Absolute option, which allows you to specify an absolute address in Fast TLB Miss Exception vector offset. Use an absolute address when the memory storing the exception handler is located outside of the processor system and subsystems of the processor system.

Fast TLB Miss Exception vector offset specifies the location of the exception vector relative to the memory module's base address. Qsys calculates the physical address of the exception vector when you modify the memory module, the offset, or the memory module's base address. In Qsys, Fast TLB Miss Exception vector displays the readonly, calculated address. The address is always a physical address, even when an MMU is present.

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 in the same design.

For information about 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 parameter editor.

Related Information[Programming Model](#)

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.

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 in the same design.

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

Related Information[Programming Model](#)

Caches and Memory Interfaces Tab

The **Caches and Memory Interfaces** tab allows you to configure the cache and tightly-coupled memory usage for the instruction and data master ports.

Table 2: Caches and Memory Interfaces Tab Parameters

Name	Description
Instruction Master	
Instruction cache	Refer to the "Instruction Master Settings" Section.
Burst transfers	
Number of tightly coupled instruction master port(s)	
Data Master	
Omit data master port	Refer to the "Data Master" Settings.
Data cache	
Data cache line size	
Burst transfers	
Data cache victim buffer implementation	
Number of tightly coupled instruction master port(s)	

The following sections describe the configuration settings available.

Related Information

- [Data Master Settings](#) on page 6
- [Instruction Master Settings](#) on page 6

Instruction Master Settings

The **Instruction Master** parameters 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.

- **Burst transfers** —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 your 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.

- **Data cache victim buffer implementation**—Specifies whether to use RAM or registers. The data cache victim buffer temporarily holds a dirty cache line while the data is written back to external memory.
- **Number of tightly coupled instruction master port(s) (Include tightly coupled instruction master port(s))**—Specifies one to four tightly-coupled instruction master ports for the Nios II processor. In Qsys, select the number from the **Number of tightly coupled instruction master port(s)** list. Tightly-coupled memory ports appear on the connection panel of the Nios II processor on the Qsys **System Contents** tab. You must connect each port to exactly one memory component in the system.

Data Master Settings

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

- **Omit data master port**—Removes the Avalon-MM data master port from the Nios II processor. The port is only successfully removed when **Data cache** is set to **None** and **Number of tightly coupled data master port(s)** is greater than zero.

Note: 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.

- **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**.
 - **Burst transfers** —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 your f_{MAX} .

Bursting is only enabled for data cache 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.
- **Number of tightly coupled data master port(s) (Include tightly coupled data master port(s))**— Specifies one to four tightly-coupled data master ports for the Nios II processor. In Qsys, select the number from the **Number of tightly coupled data master port(s)** list. Tightly-coupled memory ports appear on the connection panel of the Nios II processor on the Qsys **System Contents** tab. You must connect each port to exactly one memory component in the system.

Advanced Features Tab

The **Advanced Features** tab allows you to enable specialized features of the Nios II processor.

Table 3: Advanced Features Tab Parameters

Name	Description
General	
Interrupt controller	Refer to the "Interrupt Controller" Interfaces section.
Number of shadow register sets	Refer to the "Shadow Register Sets" section.
Include cpu_resetrequest and cpu_resettaken signals	Refer to the "Reset Signal"s section.
Assign cpuid control register value manually	Refer to the "Control Registers" section.
cpuid control register value	
Exception Checking	
Illegal instruction	Refer to the "Exception Checking" section.
Division error	
Misaligned memory access	
Extra exception information	
HardCopy Compatibility	
HardCopy compatible	Refer to the "Hardcopy Compatible" section.
ECC	

Name	Description
ECC present	Refer to the "ECC" section.

Related Information

- [ECC](#) on page 10
- [HardCopy Compatible](#) on page 10
- [Exception Checking](#) on page 8
- [Control Registers](#) on page 8
- [Reset Signals](#) on page 8
- [Shadow Register Sets](#) on page 10
- [Interrupt Controller Interfaces](#) on page 9

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 system. The signals are exported to the top level of your system.

Note: You must manually connect these signals to logic external to your Qsys system.

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

Related Information

[Processor Architecture](#)

Control Registers

The **Assign `cpuid` control register value manually** control register setting allows you to assign the `cpuid` control register value yourself. In Qsys, the automatically-assigned value is always 0x00000000, so Altera recommends always assigning the value manually.

To assign the value yourself, turn on **Assign `cpuid` control register value manually** and type a 32-bit value (in hexadecimal or decimal format) in the **`cpuid` control register value box**.

Related Information

[SOPC Builder to Qsys Migration Guidelines](#)

For information about upgrading IDs that were manually-assigned values in Qsys, refer to the SOPC Builder to Qsys Migration Guideline.

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.

Note: 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** tab **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** tab 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.

Note: When your system contains an MMU or MPU, the processor automatically generates misaligned memory access exceptions. Therefore, the **Misaligned memory access** check box is always disabled when **Include MMU** or **Include MPU** on the **Core Nios II** tab 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.

Note: 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** tab **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*.

Related Information

[Programming Model](#)

Interrupt Controller Interfaces

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.

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 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 interface, refer to “Exception Processing” in the *Programming Model* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Programming Model](#)

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.

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.

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

Related Information

[Programming Model](#)

HardCopy Compatible

The HardCopy Compatible parameter determines whether the instantiated Nios II core is compatible with HardCopy[®] devices without recompilation. This feature allows you to migrate from an FPGA device to HardCopy device without any RTL changes to the Nios II core. When **HardCopy Compatible** is on, any generated Nios II core and JTAG debug module RAM blocks are not pre-initialized.

Note: When **Device family** on the Qsys **Project Settings** tab is a HardCopy device, **HardCopy Compatible** is automatically turned on and uneditable.

Altera no longer offers HardCopy structured ASIC products for new design starts. Altera continues to support HardCopy for existing designs. Customers can find product documentation for the HardCopy structured ASIC series in the Altera mature devices product listing.

Related Information

[Altera ASICs](#)

ECC

ECC is only available for the Nios II/f core and provides ECC support for Nios II internal RAM blocks, such as instruction cache, MMU TLB, and register file. The SECDED ECC algorithm is based on Hamming codes, which detect 1 or 2 bit errors and corrects 1 bit errors. If the Nios II processor does not attempt to correct any errors and only detects them, the ECC algorithm can detect 3 bit errors.

Refer to "ECC" section in the *Nios II Core Implementation Details* chapter for more information about ECC support in the Nios II/f core.

Related Information

[Nios II Core Implementation Details](#)

MMU and MPU Settings Tab

The **MMU and MPU Settings** tab presents settings for configuring the MMU and MPU on the Nios II processor. You can select the features appropriate for your target application.

Table 4: MMU and MPU Settings Tab Parameters

Name	Description
MMU	
Process ID (PID) bits	Refer to the "MMU" section.
Optimize number of TLB entries based on device family	
TLB entries	
TLB Set-Associativity	
Micro DTLB entries	
Micro ITLB entries	
MPU	
Use Limit for region range	Refer to the "MPU" section.
Number of data regions	
Minimum data region size	
Number of instruction regions	
Minimum instruction region size	

Related Information

- [MPU](#) on page 12
- [MMU](#) on page 11

MMU

When **Include MMU** on the **Core Nios II** tab is on, the MMU settings on the **MMU and MPU Settings** tab 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 DTLB entries**—Specifies the number of entries in the micro data TLB.
- **Micro ITLB entries**—Specifies the number of entries in the micro instruction TLB.

For information about 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*.

Related Information

- [Programming Model](#)
- [Nios II Core Implementation Details](#)

MPU

When **Include MPU** on the **Core Nios II** tab is on, the MPU settings on the **MMU and MPU Settings** tab 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 MB 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 MB and must be a power of two.

Note: 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 Qsys. Maximum region size is based on the address range of slaves connected to the Nios II instruction and data masters.

For information about 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*.

Related Information

- [Programming Model](#)
- [Nios II Core Implementation Details](#)

JTAG Debug Module Tab

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

Table 5: JTAG Debug Module Tab Parameters

Name	Description
Select a Debugging Level	
Debug level	Refer to the "Debug Level Settings" section.
Include debugreq and debugack Signals	Refer to the "Debug Signals" section.
Break Vector	

Name	Description
Break vector memory	Refer to the "Break Vector" section.
Break vector offset	
Break vector	
Advanced Debug Settings	
OCI Onchip Trace	Refer to "Advanced Debug Settings" section.
Automatically generate internal 2x clock signal	

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, you might choose to reduce the JTAG debug module functionality, or remove it altogether.

Table 6: Debug Configuration Features

Feature	Description
JTAG Target Connection	Connects to the processor through the standard JTAG pins on the Altera FPGA. This connection 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 Imagination Technologies or Lauterbach GmbH.

The following sections describe the configuration settings available.

Related Information

- [Advanced Debug Settings](#) on page 15
- [Break Vector](#) on page 15

- [Debug Signals](#) on page 15
- [Debug Level Settings](#) on page 14

Debug Level Settings

The following debug levels are available in the **JTAG Debug Module** tab:

- No Debugger
- Level 1
- Level 2
- Level 3
- Level 4

The table 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 Imagination Technologies, LLC or Lauterbach GmbH.

Table 7: 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 64-KB frames ⁽³⁾	Up to 64- KB frames
Off-Chip Trace ⁽⁴⁾	0	None	None	None	128-KB frames

For information about debug features available from these third parties, search for “Nios II” on the Imagination Technologies website and the Lauterbach GmbH website.

Related Information

- [Lauterbach GmbH](#)
- [Imagination Technologies, LLC](#)

⁽¹⁾ Level 4 requires the purchase of a software upgrade from Imagination Technologies or Lauterbach.

⁽²⁾ Not including the dedicated JTAG pins on the Altera FPGA.

⁽³⁾ An additional license from Imagination Technologies is required to use more than 16 frames.

⁽⁴⁾ Off-chip trace requires the purchase of additional hardware from Imagination Technologies or Lauterbach.

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 Qsys system.

For more information about the debug signals, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Processor Architecture](#)

Break Vector

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

When the Nios II processor does not contain a JTAG debug module, you can edit the break vector parameters in the manner described in "General Exception Vector" section.

Related Information

[General Exception Vector](#) on page 3

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.

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

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 system manually.

For more information about trace frames, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Related Information

[Processor Architecture](#)

Custom Instruction Tab

In Qsys, custom instructions are components in your design that you manually connect to the processor in the Qsys **System Contents** tab. Existing custom instruction components are available on the **Component Library** tab under **Custom Instruction Modules**. Thus, the **Custom Instruction** tab in the Nios II Processor parameter editor is not used in Qsys.

To create your own custom instruction using the component editor, click **New Component** on the File menu in Qsys. After finishing in the component editor, the new instruction appears on the **Component Library** tab under **Custom Instruction Modules** in Qsys.

Note: 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_datab`, `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*.

Related Information

[Nios II Custom Instruction User Guide](#)

Altera-Provided Custom Instructions

The following sections describe the custom instructions Altera provides.

Note: The Endian Converter Custom Instruction and Interrupt Vector Custom Instruction are not available in Qsys.

For information about converting SOPC Builder designs to Qsys, refer to the *SOPC Builder to Qsys Migration Guidelines*.

Related Information

[SOPC Builder to Qsys Migration Guidelines](#)

Floating Point Hardware 2 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 Floating Point Hardware 2 Custom Instruction is a high performance component with predefined custom instructions that implement single-precision floating-point operations. This component offers improved performance with lower cycle counts for addition, subtraction, multiplication and division, and also supports floating-point operations such as square root, comparison, minimum/maximum, negate/absolute, and conversion.

The Floating Point Hardware 2 component is composed of two custom instructions:

- Combinational custom instruction—Implements the minimum, maximum, compare, negate, and absolute operations.
- Multi-cycle custom instruction—Implements the add, subtract, multiply, divide, square root, and convert operations.

The component has two slaves, one slave for the combinatorial custom instruction and the other slave for the multi-cycle custom instruction.

The opcode extensions for the Floating Point Hardware 2 custom instructions are 224 through 255. Refer to the Floating Point Custom Instruction 2 Operation Summary table in the "Floating Point Custom Instruction 2 Component" section in the *Processor Architecture* chapter for details.

To add the Floating Point Hardware 2 custom instruction to the Nios II processor in Qsys, select **Floating Point Hardware 2** under **Embedded Processors** in the **Component Library** tab. Connect the two slave interfaces to the Nios II custom instruction master.

Related Information**Floating Point Custom Instruction 2 Component****Floating-Point Hardware Custom Instruction**

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.

Note: 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 in Qsys, select **Floating Point Hardware** under **Custom Instruction Modules** on the **Component Library** tab, and click **Add**. By default, Qsys includes floating-point addition, subtraction, and multiplication, but omit the more resource intensive floating-point division. The **Floating Point Hardware** parameter editor appears, giving you the option to include the floating-point division hardware.

Table 8: Floating Point Hardware Parameters

Name	Values	Description
Use floating point division hardware	On/Off	Specifies inclusion of floating-point division hardware.

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 more information about the floating-point custom instructions, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Related Information**Processor Architecture****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 in Qsys, select **Bitswap** under **Custom Instruction Modules** on the **Component Library** tab, 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*.

Related Information

[Nios II Custom Instruction User Guide](#)

The Quartus II IP File

The Quartus[®] II IP file (**.qip**) is a file generated by the MegaWizard[™] Plug-In Manager, 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 Qsys system. However, some complex components generate a separate **.qip** file, so the system **.qip** file references the component **.qip** file.

Document Revision History

Table 9: Document Revision History

Date	Version	Changes
April 2015	2015.04.02	Maintenance release.
February 2014	13.1.0	<ul style="list-style-type: none"> Added information about the Floating Point Custom Instruction 2 Component Added information about ECC support. Removed references to SOPC Builder.
May 2011	11.0.0	<ul style="list-style-type: none"> Revised the entire chapter for the new Qsys system integration tool. Replaced GUI screen shots with parameter tables. Incorporated interrupt vector custom instruction information from the <i>Processor Architecture</i> chapter.
December 2010	10.1.0	Maintenance release.
July 2010	10.0.0	Maintenance release.
November 2009	9.1.0	<ul style="list-style-type: none"> Added external interrupt controller interface information. Added shadow register set information.
March 2009	9.0.0	Maintenance release.
November 2008	8.1.0	<ul style="list-style-type: none"> Added <code>debugreq</code> and <code>debugack</code> signal options to Advanced Features tab. Added <code>cpuid</code> manual override options to Advanced Features tab.
May 2008	8.0.0	<ul style="list-style-type: none"> Added MMU options to Nios II Core and Advanced Features tabs. Added exception handling options Advanced Features tab.

Date	Version	Changes
October 2007	7.2.0	Changed title to match other Altera documentation.
May 2007	7.1.0	<ul style="list-style-type: none">Revised to reflect new MegaWizard interface.Added Endian Converter Custom Instruction and Bitswap Custom Instruction section.Added table of contents to Introduction section.Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	<ul style="list-style-type: none">Add section on interrupt vector custom instruction.Add section on system-dependent Nios II processor settings.
May 2006	6.0.0	<ul style="list-style-type: none">Added details on floating-point custom instructions.Added section on Advanced Features tab.
October 2005	5.1.0	Maintenance release.
May 2005	5.0.0	<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	1.1	<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	1.0	Initial release.