Explore more resources
Altera® Design Hub

altera
An Intel Company

Nios® V Processor Reference Manual

Updated for Quartus® Prime Design Suite: **25.1**

Online Version
Send Feedback

683632
2025.05.25

Contents

1. Overview **4**
1.1. Quartus® Prime Software Support 5
2. Nios V/c Processor **6**
2.1. Processor Performance Benchmarks 6
2.2. Processor Architecture 7
2.2.1. General-Purpose Register File 8
2.2.2. Arithmetic Logic Unit 9
2.2.3. Reset Signals 10
2.2.4. Memory and I/O Organization 10
2.2.5. Error Correction Code (ECC) 14
2.3. Programming Model 15
2.3.1. Privilege Levels 15
2.4. Core Implementation 15
2.4.1. Instruction Set Reference 15
3. Nios V/m Processor **17**
3.1. Processor Performance Benchmarks 17
3.1.1. Pipelined 17
3.1.2. Non-pipelined 19
3.2. Processor Pipeline 20
3.2.1. Pipelined Architecture 20
3.2.2. Non-pipelined Architecture 21
3.3. Processor Architecture 21
3.3.1. General-Purpose Register File 22
3.3.2. Arithmetic Logic Unit 23
3.3.3. Reset and Debug Signals 24
3.3.4. Control and Status Registers 25
3.3.5. Instruction Cycles 25
3.3.6. Trap Controller (CLINT) 26
3.3.7. Memory and I/O Organization 32
3.3.8. RISC-V based Debug Module 36
3.3.9. Error Correction Code (ECC) 50
3.4. Programming Model 51
3.4.1. Privilege Levels 51
3.4.2. Control and Status Registers (CSR) Mapping 51
3.5. Core Implementation 58
3.5.1. Instruction Set Reference 58
4. Nios V/g Processor **60**
4.1. Processor Performance Benchmarks 60
4.2. Processor Pipeline 61
4.3. Processor Architecture 62
4.3.1. General-Purpose Register File 63
4.3.2. Shadow Register 65
4.3.3. Arithmetic Logic Unit 72
4.3.4. Multipy and Divide Units 73
4.3.5. Floating-Point Unit 73

4.3.6. Custom Instruction..... 77
4.3.7. Instruction Cycles..... 77
4.3.8. Reset and Debug Signals..... 78
4.3.9. Control and Status Registers 79
4.3.10. Trap Controller (CLINT)..... 80
4.3.11. Trap Controller (CLIC)..... 86
4.3.12. Memory and I/O Organization..... 101
4.3.13. RISC-V based Debug Module..... 111
4.3.14. Error Correction Code (ECC)..... 125
4.3.15. Branch Prediction..... 128
4.3.16. Lockstep Module..... 128
4.4. Programming Model..... 128
4.4.1. Privilege Levels..... 128
4.4.2. Control and Status Registers (CSR) Mapping (CLINT)..... 129
4.4.3. Control and Status Registers (CSR) Mapping (CLIC)..... 136
4.5. Core Implementation..... 146
4.5.1. Instruction Set Reference..... 146

5. Nios V Processor Reference Manual Archives..... 149

6. Document Revision History for the Nios V Processor Reference Manual..... 150

1. Overview

The Nios® V processor is a soft processor core with the following characteristics:

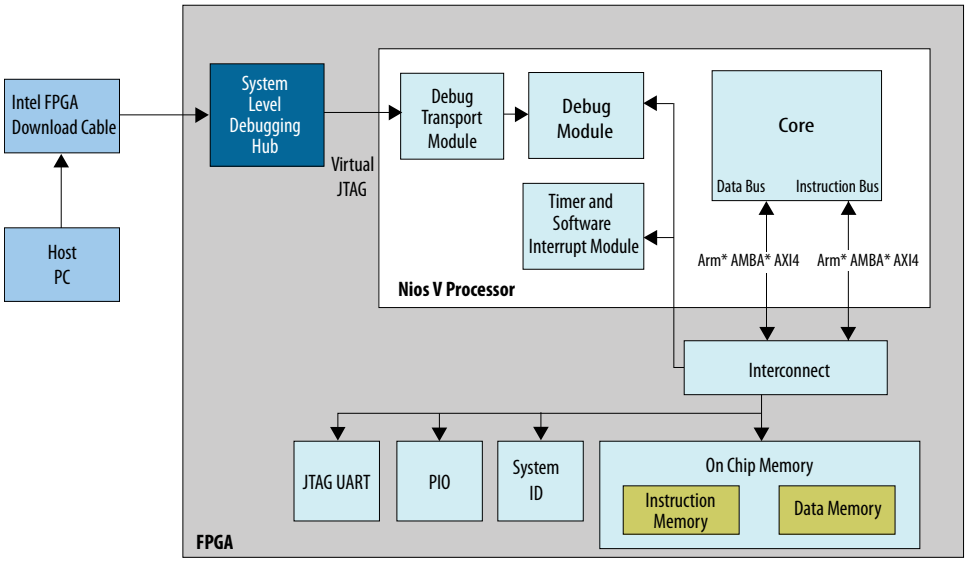
- Designed for Altera FPGA devices and based on RISC-V* specification
- Does not include peripherals or the connection logic to external devices
- Includes only the circuitry required to implement the Nios V processor architecture

The following table lists the available variants:

Table 1. Nios V Processor Variants

Processor Core	Variant
Nios V/c	Compact Microcontroller
Nios V/m	Microcontroller
Nios V/g	General-purpose processor

Figure 1. Nios V Processor System on Altera FPGA



Related Information

- RISC-V® Specifications
- Nios V/c Processor on page 6
  More information about the Nios V/c processor (Compact microcontroller)
- Nios V/m Processor on page 17
  More information about the Nios V/m processor (Microcontroller).

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Nios V/g Processor on page 60
  More information about the Nios V/g processor (General-purpose processor).

1.1. Quartus® Prime Software Support

Nios V processor build flow is different for Quartus® Prime Pro Edition software and Quartus Prime Standard Edition software. Refer to *AN 980: Nios V Processor Quartus Prime Software Support* for more information about the differences.

Related Information

AN 980: Nios V Processor Quartus Prime Software Support

2. Nios V/c Processor

The Nios V/c processor is a compact microcontroller core developed by Altera based on the RISC-V RV32I instruction set. It supports the functional units described in this document.

Related Information

Overview on page 4

2.1. Processor Performance Benchmarks

Table 2. Nios V/c Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software

Quartus Prime Edition	FPGA Used	OPN	f_{MAX} (MHz)	Logic Size	Architecture Performance	
					DMIPS/MHz Ratio	CoreMark/MHz Ratio
Quartus Prime Pro Edition	Cyclone® 10	10CX220YF780I5G	260	408 ALM	0.227	0.17
	Arria® 10	10AS066N3F40E2SG	281	405 ALM		
	Stratix® 10	1SX280LU2F50E2VG	315	473 ALM		
	Agilex™ 7	AGFB014R24AR0	358	447 ALM		
	Agilex 5	A5EC065BB32AE4S	311	435 ALM		
Quartus Prime Standard Edition	Cyclone IV E	EP4CE115F29I8L	118	1022 LE	0.268	0.201
	Cyclone V	5CGTFD9E5F35C7	155	423 ALM		
	Arria V	5AGXMB7G6F35C6	175	414 ALM		
	Arria V GZ	5AGZME7K2F40C3	289	372 ALM		
	Stratix V	5SGXEA7K2F40C2	332	372 ALM		
	Cyclone 10 LP	10CL120YF780I7G	137	1025 LE		
	Arria 10	10AS066N3F40E2SG	325	355 ALM		
	MAX® 10	10M50DAF484C7G	137	1022 LE		

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Table 3. Benchmark Parameters for Quartus Prime Software

Parameter		Settings/Description	
		Quartus Prime Pro Edition	Quartus Prime Standard Edition
Quartus Prime seed		Maximum performance result are based on 10 seed sweep from Quartus Prime Pro Edition software version 25.1.	Maximum performance result are based on 10 seed sweep from Quartus Prime Standard Edition software version 23.1.
Device speed grade		Fastest speed grade from each Altera FPGA device family.	
Defined peripherals		• Nios V/c processor core (without debug module and internal timer). • 128 KB on-chip memory for the instruction and data bus. • JTAG UART Intel® FPGA IP. • Interval Timer Core.	
Toolchain	Version	• riscv32-unknown-elf-gcc (GCC) version 13.2.0 • CMake Version: 3.31.3	• riscv32-unknown-elf-gcc (GCC) version 12.1.0 • CMake Version: 3.27.1
	Compiler configuration	• Compiler flags: `-O3` • Assembler options: `-Wa -gdwarf2` • Compile options: `-Wall -Wformat-security -march=rv32i -mabi=ilp32`	

Altera® uses the same Quartus Prime design example for maximum performance benchmark(fMAX) and logic size benchmarks. The compiler settings are:

- **Superior Performance with Maximum Placement Effort** in Quartus Prime Pro Edition software.
- **High Performance Effort** in Quartus Prime Standard Edition software.

Note: Results may vary depending on the version of the Quartus Prime software, the version of the Nios V processor, compiler version, target device and the configuration of the processor. Additionally, any changes to the system logic design can change the performance and LE usage. All results are generated from design built with Platform Designer.

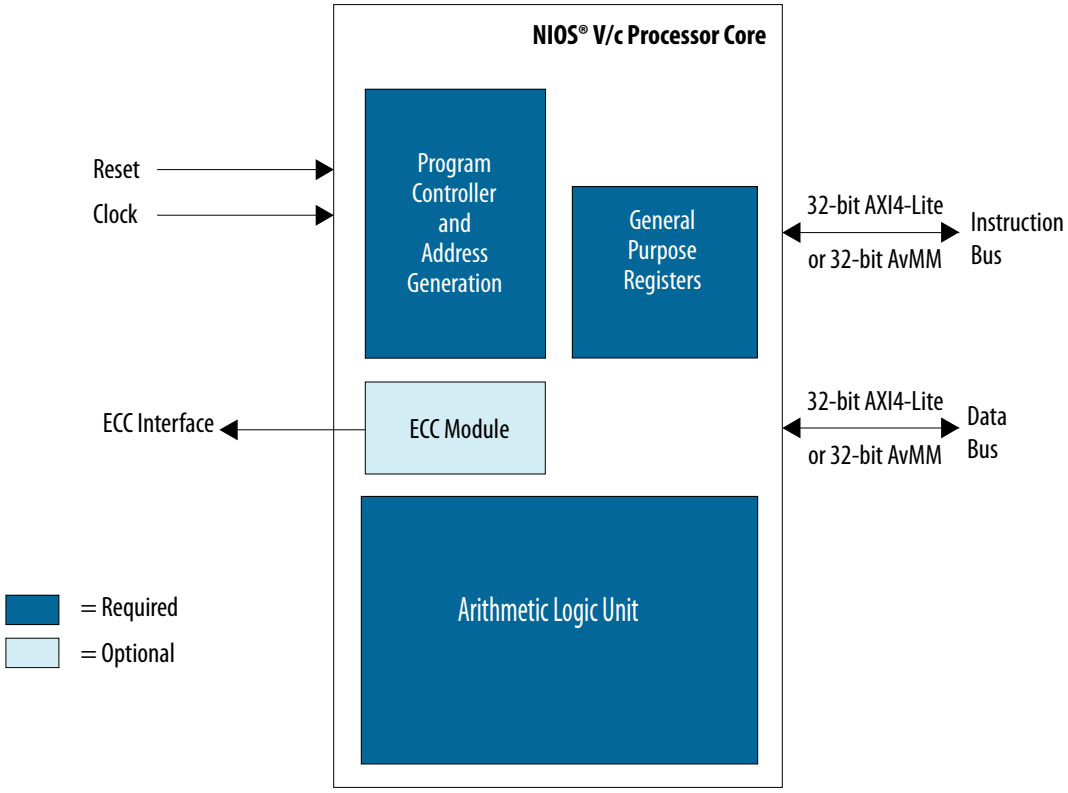
2.2. Processor Architecture

The Nios V/c processor architecture implements the RV32I instruction-set architecture (ISA).

The Nios V/c processor architecture defines the following functional units:

- General-purpose register file
- Arithmetic logic unit (ALU)
- Instruction and Data Bus
- ECC module

Figure 2. Nios V/c Processor Core Block Diagram



2.2.1. General-Purpose Register File

Nios V/c processor implementation supports a flat register file. The register file contains thirty-two 32-bit general-purpose integer registers. Nios V/c processor implements the general-purpose register using M20K memories, which do not support two read ports. Hence, Nios V/c processor duplicates the register files so that two different source registers for an instruction are available in a single cycle. After performing ALU operations, the processor core writes the same result to the destination register in both memories.

Table 4. Nios V Processor General-Purpose Register File

Register	ABI	Description
x0	zero	Hardwired to zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary 0
x6	t1	Temporary 1
x7	t2	Temporary 2

continued...

Register	ABI	Description
x8	s0/fp	• Saved register 0 • Frame pointer
x9	s1	Saved register 1
x10	a0	• Function argument 0 • Return value 0
x11	a1	• Function argument 1 • Return value 1
x12	a2	Function argument 2
x13	a3	Function argument 3
x14	a4	Function argument 4
x15	a5	Function argument 5
x16	a6	Function argument 6
x17	a7	Function argument 7
x18	s2	Saved register 2
x19	s3	Saved register 3
x20	s4	Saved register 4
x21	s5	Saved register 5
x22	s6	Saved register 6
x23	s7	Saved register 7
x24	s8	Saved register 8
x25	s9	Saved register 9
x26	s10	Saved register 10
x27	s11	Saved register 11
x28	t3	Temporary 3
x29	t4	Temporary 4
x30	t5	Temporary 5
x31	t6	Temporary 6

2.2.2. Arithmetic Logic Unit

The arithmetic logic unit (ALU) operates on data stored in general-purpose registers. ALU operations take one or two inputs from registers and store the result back into the register.

Table 5. Fundamental Data Operations of the ALU

Category	Description
Arithmetic	Addition and subtraction on signed and unsigned operands.
Relational	Equal, not-equal, greater-than-or-equal, and less-than relational operations (==, != >=, <).
Logical	AND, OR, NOR, and XOR logical operations.
Shift	Logical and arithmetic shift operations.

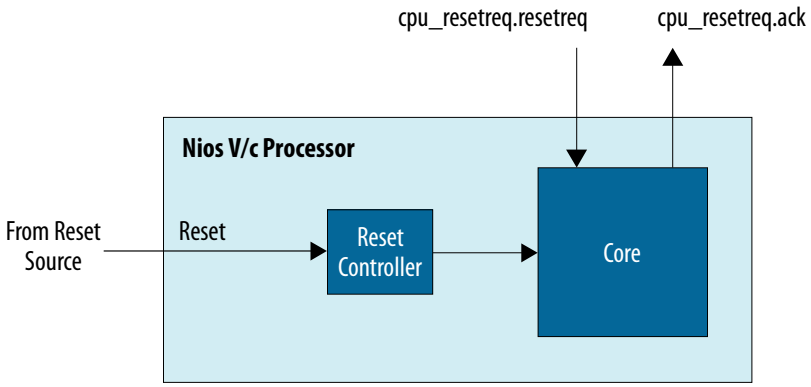
For load and store instructions, the Nios V/c processor uses the ALU to calculate the memory address. For conditional control transfer instructions, Nios V/c processor uses the relational operations in the ALU to determine if the processor takes or leaves the branch.

2.2.3. Reset Signals

Table 6. Reset Signals

Interface	Type	Description
`reset`	Reset	A global hardware reset input signal that forces the Nios V processor to reset immediately.
`cpu_resetreq`	Conduit	An optional local reset ports which appear after you enable **Add Reset Request Interface** parameter. The signal consists of an input `resetreq` signal and an output `ack` signal that trigger the Nios V processor to reset without affecting other components in a Nios V processor system. • You can request a reset to the Nios V processor core by asserting the `resetreq` signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the ack signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

Figure 3. Nios V/c Processor Reset Network



2.2.4. Memory and I/O Organization

You can configure the Nios V/c processor systems. Consequently, the memory and I/O organization varies from system to system. A Nios V/c processor core uses one or more of the following ports to provide access to memory and I/O:

- Instruction manager port: A Memory-Mapped manager port that connects to instruction memory via system interconnect fabric.
- Data manager port: A Memory-Mapped manager port that connects to data memory and peripherals via the system interconnect fabric.

Nios V/c Processor Core Memory Mapped I/O Access: Both data memory and peripherals are mapped into the address space of the data manager port. Nios V/c processor core uses little-endian byte ordering. Words and half-words are stored in memory with the more-significant bytes at higher addresses. The Nios V/c processor core does not specify anything about the existence of memory and peripherals. The quantity, type, and connection of memory and peripherals are system dependent.

2.2.4.1. Instruction and Data Buses

2.2.4.1.1. Instruction Manager Port

Nios V/c processor instruction bus is implemented as a 32-bit Arm* Advanced Microcontroller Bus Architecture (AMBA*) 4 AXI-Lite manager port as default interface. You can switch it to Avalon® memory-mapped interface for smaller logic utilization.

The instruction manager port:

- Performs a single function: it fetches instructions to be executed by the processor.
- Does not perform any write operations.
- Issue another read request after the current request is completed.
- Always retrieves 32-bit of data. Every instruction fetch returns a full instruction word, regardless of the width of the target memory. The widths of memory in the Nios V/c processor system is not applicable to the programs. Instruction address is always aligned to a 32-bit word boundary.
- Does not require any burst adapter because it is non-bursting.

Table 7. AMBA* 4 AXI-Lite Instruction Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Unused	[31:0]	Output
	awprot	Unused	[2:0]	Output
	awvalid	Unused	1	Output
	awready	Unused	1	Input
Write Data Channel	wdata	Unused	[31:0]	Output
	wstrb	Unused	[3:0]	Output
	wvalid	Unused	1	Output
	wready	Unused	1	Input
Write Response Channel	bresp	Unused	[1:0]	Input
	bvalid	Unused	1	Input
	bready	Unused	1	Output
Read Address Channel	araddr	Instruction Address (Program Counter)	[31:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Instruction address valid	1	Output
	arready	Instruction address ready	1	Input

continued...

Interface	Signal	Role	Width	Direction
		(from memory)		
Read Data Channel	rdata	Instruction	[31:0]	Input
	rresp	Instruction response: Non-zero value denotes instruction access fault exception	[1:0]	Input
	rvalid	Instruction valid	1	Input
	rready	Constant 1	1	Output

Table 8. Avalon Memory-Mapped Instruction Interface Signals

Signal	Role	Width	Direction
readdata	Instruction	[31:0]	Agent → Processor
waitrequest	Force the processor to wait until the agent is ready	1	Agent → Processor
readdatavalid	Instruction is valid	1	Agent → Processor
response	Instruction response: Non-zero value denotes instruction access fault exception	[31:0]	Agent → Processor
address	Instruction Address (Program Counter)	[31:0]	Processor → Agent
read	Asserted to indicate a read transfer.	1	Processor → Agent

2.2.4.1.2. Data Manager Port

The Nios V/c processor data bus is implemented as a 32-bit AMBA* 4 AXI-Lite manager port as default interface. You can switch it to Avalon memory-mapped interface for smaller logic utilization.

The data manager port:

- Performs read data from memory or a peripheral when the processor executes a load instruction.
- Performs write data to memory or a peripheral when the processor executes a store instruction.
- Does not require any burst adapter because it is non-bursting.

Address on `axaddr` signal is always aligned to size of the transfer. For store instructions, respective writes strobe bits are asserted to indicate bytes being written.

Nios V/c processor core does not support speculative issue of load/store instruction. Hence, the core issues only one load or store instruction and waits until the issued instruction is complete.

Table 9. AMBA* 4 AXI-Lite Data Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Store address	[31:0]	Output
	awprot	Unused	[2:0]	Output
	awvalid	Store address valid	1	Output

continued...

Interface	Signal	Role	Width	Direction
	awready	Store address ready (from memory)	1	Input
Write Data Channel	wdata	Store data	[31:0]	Output
	wstrb	Byte position in word	[3:0]	Output
	wvalid	Store data valid	1	Output
	wready	Store data ready (from memory)	1	Input
Write Response Channel	bresp	Store response: Non-zero value denotes store access fault exception.	[1:0]	Input
	bvalid	Store response valid	1	Input
	bready	Constant 1	1	Output
Read Address Channel	araddr	Load address	[31:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Load address valid	1	Output
	arready	Load address ready (from subordinates)	1	Input
Read Data Channel	rdata	Load data	[31:0]	Input
	rresp	Load response: Non-zero value denotes load access fault exception	[1:0]	Input
	rvalid	Load data valid	1	Input
	rready	Constant 1	1	Output

Table 10. Avalon Memory-Mapped Data Interface Signals

Signal	Role	Width	Direction
readdata	Load data	[31:0]	Agent → Processor
waitrequest	Force the processor to wait until the agent is ready	1	Agent → Processor
readdatavalid	Load data is valid	1	Agent → Processor
response	Load/Store response: Non-zero value denotes access fault exception	[31:0]	Agent → Processor
address	Load/Store address	[31:0]	Processor → Agent
read	Asserted to indicate a read transfer	1	Processor → Agent
write	Asserted to indicate a write transfer	1	Processor → Agent
writedata	Store data	[31:0]	Processor → Agent
byteenable	Indicates the load/store instruction size • Byte (LB/SB), • Halfword (LH/SH), or • Word (LW/SW)	[3:0]	Processor → Agent
writeresponsevalid	Asserted to indicate a valid write response	1	Agent → Processor

2.2.4.1.3. Choosing a Suitable Interface

The Nios V/c processor core offers an Avalon memory-mapped and Advanced eXtensible Interface (AXI) interface for both the instruction manager port and the data manager port. You can choose one of the interfaces in Platform Designer based on your design.

For example, if most IPs in Platform Designer use Avalon memory-mapped interface, you can choose the Avalon memory-mapped interface in the Nios V/c processor core as the bridge interface. Similarly, if most IPs in Platform Designer use AXI Interface, you can choose the AXI Interface in the Nios V/c processor core as the bridge interface.

The benefits of using the appropriate interface are as follows:

- Reduces the amount of bridging logic
- Reduces overhead bandwidth
- Provides better performance

2.2.4.2. Address Map

The address map for memories and peripherals in a Nios V/c processor system is design dependent. The Reset Address is part of the processor. You can specify the Reset Address in Platform Designer during system configuration.

2.2.5. Error Correction Code (ECC)

The Nios V/c processor core has the option to enable error detection and ECC status reporting for the RAM block, that is the Register file. Each RAM block has its own source ID. When an ECC event occurs, the processor transmits the source ID and ECC status to the ECC interface.

- If the ECC event is a correctable error, the processor continues to operate after correcting the error. The correction made is not written back to its memory source.
- If the ECC event is an un-correctable error, the processor continues to operate without correcting the error. Thus, it is operating under corrupted data. You need to reset either the processor core alone or the entire system once `cpu_ecc_status` reports Uncorrectable ECC error.

Note: To reset only the processor core, apply the **Reset Request Interface** to safely reset the Nios V processor (cleared of any outstanding operations). To reset the entire system, you can use the hard reset interface instead.

The ECC interface allows external logic to monitor ECC errors from the Nios V/c processor. The interface is a conduit, made up of the following output signals.

- `cpu_ecc_status` : Indicates the error status
- `cpu_ecc_source` : Indicates the error source.

Table 11. cpu_ecc_status

2-bits Encoding	Description	Effects on Software
2'b00	No ECC event	None
2'b01	Reserved	Not Applicable
2'b10	Correctable single bit ECC error	None
2'b11	Un-correctable ECC error	Likely fatal to the processor.

Table 12. cpu_ecc_source

4-bits Encoding	ECC Source	Available
4'b0000	No ECC event	Always
4'b0001	General Purpose Register (GPR)	Always
4'b0010 ~ 4'b1110	Other RAM Blocks	Not Available
4'b1111	Reserved	Not Applicable

Related Information

Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide

2.3. Programming Model

2.3.1. Privilege Levels

The privilege levels in Nios V/c processor are designed based on the RISC-V architecture specification. The Nios V/c processor only supports Machine Mode (M-mode), which provides low-level access to the machine implementation.

Related Information

RISC-V® Specifications

2.4. Core Implementation

2.4.1. Instruction Set Reference

The Nios V/c processor is based on the RISC-V RV32I specification. RV32I is a base integer instruction set. RV321 supports 6 types of instruction format. They are R-type, I-type, S-type, B-type, U-type, and J-type.

Table 13. Instruction Formats (R-type)

Bit Field (R-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
funct7							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd			opcode								

Table 14. Instruction Formats (I-type)

Bit Field (I-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:0]												rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd					opcode						

Table 15. Instruction Formats (S-type)

Bit Field (S-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:5]							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:0]					opcode						

Table 16. Instruction Formats (B-type)

Bit Field (B-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[12]	imm[10:5]						rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:1]			imm[11]		opcode						

Table 17. Instruction Formats (U-type)

Bit Field (U-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[31:16]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Table 18. Instruction Formats (J-type)

Bit Field (J-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[20]	imm[10:1]										imm[11]	imm[19:16]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Related Information

RISC-V® Specifications
More information about the instruction set of the RV32I Base Integer.

3. Nios V/m Processor

The Nios V/m processor is a microcontroller core developed by Intel based on the RISC-V instruction set and supports the functional units described in this document.

The Nios V/m processor supports two distinct configurations:

- Pipelined
  - Implements RV32I_Zicsr instruction set.
  - Supports five-stages pipelined datapath.
  - Supports CLINT-Direct and CLINT-Vectored interrupt mode.
- Non-pipelined
  - Implements RV32I_Zicsr instruction set.
  - Supports non-pipelined datapath.
  - Supports CLINT-Direct interrupt mode only.

Related Information

Overview on page 4

3.1. Processor Performance Benchmarks

3.1.1. Pipelined

Table 19. Nios V/m Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software

Quartus Prime Edition	FPGA Used	OPN	fMAX (MHz)	Logic Size	Architecture Performance	
					DMIPS/MHz Ratio	CoreMark/MHz Ratio
Quartus Prime Pro Edition	Cyclone 10	10CX220YF780I5G	285	1116 ALM	0.63	0.49
	Arria 10	10AS066N3F40E2SG	298	1121 ALM		
	Stratix 10	1SX280LU2F50E2VG	327	1283 ALM		
	Agilex 7	AGFB014R24AR0	422	1285 ALM		
	Agilex 5	A5EC065BB32AE4S	339	1246 ALM		
Quartus Prime Standard Edition	Cyclone IV E	EP4CE115F29I8L	107	2831 LE	0.650	0.441
	Cyclone V	5CGTFD9E5F35C7	145	1253 ALM		
	Arria V	5AGXMB7G6F35C6	146	1222 ALM		
	Arria V GZ	5AGZME7K2F40C3	264	1215 ALM		

continued...

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Quartus Prime Edition	FPGA Used	OPN	fMAX (MHz)	Logic Size	Architecture Performance	
					DMIPS/MHz Ratio	CoreMark/MHz Ratio
	Stratix V	5SGXEA7K2F40C2	304	1199 ALM		
	Cyclone 10 LP	10CL120YF780I7G	135	2848 LE		
	Arria 10	10AS066N3F40E2SG	290	1132 ALM		
	MAX 10	10M50DAF484C7G	123	2864 LE		

Table 20. Benchmark Parameters for Quartus Prime Software

Parameter		Settings/Description	
		Quartus Prime Pro Edition	Quartus Prime Standard Edition
Quartus Prime seed		Maximum performance result are based on 10 seed sweep from Quartus Prime Pro Edition software version 25.1.	Maximum performance result are based on 10 seed sweep from Quartus Prime Standard Edition software version 23.1.
Device speed grade		Fastest speed grade from each Altera FPGA device family.	
Defined peripherals		• Nios V/m processor core (without debug module and internal timer).	• Nios V/m processor core (without debug module).
		• 128 KB on-chip memory for the instruction and data bus. • JTAG UART Intel FPGA IP. • Interval Timer Core.	
Toolchain	Version	• riscv32-unknown-elf-gcc (GCC) version 13.2.0 • CMake Version: 3.31.3	• riscv32-unknown-elf-gcc (GCC) version 12.1.0 • CMake Version: 3.27.1
	Compiler configuration	• Compiler flags: `-03` • Assembler options: `-Wa -gdwarf2` • Compile options: `-Wall -Wformat-security -march=rv32i -mabi=ilp32`	

Altera uses the same Quartus Prime design example for maximum performance benchmark(fMAX) and logic size benchmarks. The compiler settings are:

- **Superior Performance with Maximum Placement Effort** in Quartus Prime Pro Edition software.
- **High Performance Effort** in Quartus Prime Standard Edition software.

Note: Results may vary depending on the version of the Quartus Prime software, the version of the Nios V processor, compiler version, target device and the configuration of the processor. Additionally, any changes to the system logic design might change the performance and LE usage. All results are generated from design built with Platform Designer.

3.1.2. Non-pipelined

Table 21. Nios V/m Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software

Quartus Prime Edition	FPGA Used	OPN	f_{MAX} (MHz)	Logic Size	Architecture Performance	
					DMIPS/MHz Ratio	CoreMark/MHz Ratio
Quartus Prime Pro Edition	Cyclone 10	10CX220YF780I5G	313	752 ALM	0.227	0.170
	Arria 10	10AS066N3F40E2SG	315	736 ALM		
	Stratix 10	1SX280LU2F50E2VG	357	837 ALM		
	Agilex 7	AGFB014R24AR0	414	817 ALM		
	Agilex 5	A5EC065BB32AE4S	381	821 ALM		
Quartus Prime Standard Edition	Cyclone IV E	EP4CE115F29I8L	117	1598 LE	0.268	0.201
	Cyclone V	5CGTFD9E5F35C7	144	705 ALM		
	Arria V	5AGXMB7G6F35C6	159	708 ALM		
	Arria V GZ	AGZME7K2F40C3	281	658 ALM		
	Stratix V	5SGXEA7K2F40C2	330	641 ALM		
	Cyclone 10 LP	10CL120YF780I7G	135	1604 LE		
	Arria 10	10AS066N3F40E2SG	316	559 ALM		
	MAX 10	10M50DAF484C7G	127	1619 LE		

Table 22. Benchmark Parameters for Quartus Prime Software

Parameter		Settings/Description	
		Quartus Prime Pro Edition	Quartus Prime Standard Edition
Quartus Prime seed		Maximum performance result are based on 10 seed sweep from Quartus Prime Pro Edition software version 25.1.	Maximum performance result are based on 10 seed sweep from Quartus Prime Standard Edition software version 23.1.
Device speed grade		Fastest speed grade from each Altera FPGA device family.	
Defined peripherals		• Nios V/m processor core (without debug module and internal timer). • 128 KB on-chip memory for the instruction and data bus. • JTAG UART Intel FPGA IP. • Interval Timer Core	
Toolchain	Version	• riscv32-unknown-elf-gcc (GCC) version 13.2.0 • CMake Version: 3.31.3	• riscv32-unknown-elf-gcc (GCC) version 12.1.0 • CMake Version: 3.27.1
	Compiler configuration	• Compiler flags: `-O3` • Assembler options: `-Wa -gdwarf2` • Compile options: `-Wall -Wformat-security -march=rv32i -mabi=ilp32`	

Altera uses the same Quartus Prime design example for maximum performance benchmark(fMAX) and logic size benchmarks. However, the compiler settings are different for each benchmarks:

- **Superior Performance with Maximum Placement Effort** in Quartus Prime Pro Edition software.
- **High Performance Effort** in Quartus Prime Standard Edition software.

Note: Results may vary depending on the version of the Quartus Prime software, the version of the Nios V processor, compiler version, target device and the configuration of the processor. Additionally, any changes to the system logic design might change the performance and LE usage. All results are generated from design built with Platform Designer.

3.2. Processor Pipeline

3.2.1. Pipelined Architecture

The Nios V/m processor employs a five-stage datapath.

Table 23. Processor Pipeline Stages

Stage	Denotation	Function
F	Instruction fetch	• PC+4 calculation • Next instruction fetch • Pre-decode for register file read
D	Instruction decode	• Decode the instruction • Register file read data available • Hazard resolution and data forwarding
E	Instruction execute	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
M	Memory	• Memory and multicycle operations • Register file write • Branch redirection
W	Write back	• Facilitates data dependency resolution by providing general-purpose register value.

The Nios V/m processor implements the general-purpose register file using the M20K memory blocks. The processor takes one cycle to read from an M20K location. Therefore, the F-stage initiates register file reads so general-purpose register values are available in D-stage.

Writing to the M20K location takes two cycles. Therefore, the M-stage initiates writes to a general-purpose register. If there is a dependency to resolve, the M-stage carries forward the value to the W-stage.

The core resolves data dependencies in the D-stage. Operands can move from register file read or E-stage, M-stage, or W-stage.

Reasons for the pipeline stalling:

- Data dependency—if the source operand is not available in the D-stage, instruction in the D-stage and F-stage stalls until the operand becomes available. This happens when the destination general-purpose register of a load or multicycle instruction in the E-stage or M-stage is the source for the instruction in the D-stage.
- Resource stall—if a memory operation or multicycle is pending in the M-stage, the instructions in the preceding stages stall until M-stage completes the instruction.

3.2.2. Non-pipelined Architecture

The Nios V/m processor supports a non-pipelined datapath.

Table 24. Processor Non-pipelined Stages

Stage	Denotation	Function
F	Instruction fetch	Pre-decode for register file read
D	Instruction decode	• Decode the instruction • Register file read data available • Hazard resolution and data forwarding
E	Instruction execute	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
M	Memory	• Memory and multicycle operations • Register file write • Next PC logic • Branch redirection

The Nios V/m processor implements the general-purpose register file using the M20K memory blocks. The processor takes one clock cycle to read from an M20K location. Therefore, the F-stage initiates register file reads so general-purpose register values are available in D-stage.

One instruction is available in the processor datapath at any time. Instructions flow from F-stage to M-stages without any stalls. Instruction and associated control logic are registered during D-stage, E-stage, and M-stage.

The processor requests the next instruction during the M-stage.

- For single cycle instructions, the processor makes the request as soon as the single cycle instruction enters M-stage.
- For multicycle instructions, the processor makes the request as soon as the multicycle instruction completes.

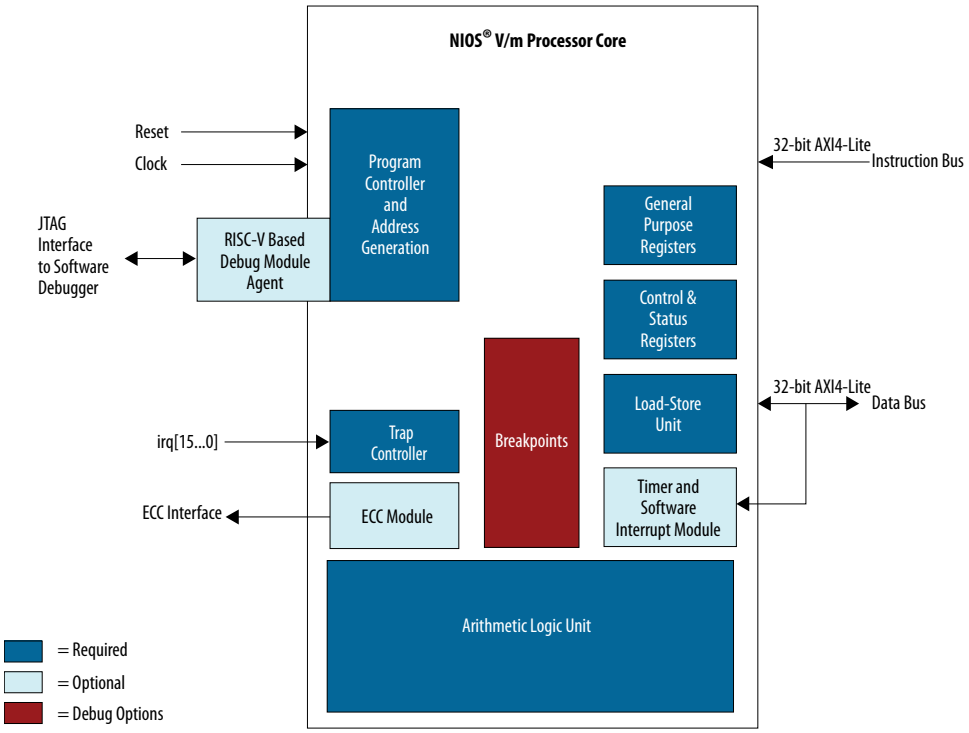
3.3. Processor Architecture

The Nios V/m processor architecture describes an instruction-set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions.

The Nios V/m processor architecture defines the following functional units:

- General-purpose register file
- Arithmetic logic unit (ALU)
- Control and status registers (CSR)
- Trap controller
- Instruction bus
- Data bus
- RISC-V based debug module
- ECC module

Figure 4. Nios V/m Processor Core Block Diagram



3.3.1. General-Purpose Register File

Nios V/m processor implementation supports a flat register file. The register file contains thirty-two 32-bit general-purpose integer registers. Nios V/m processor implements the general-purpose register using M20K memories, which do not support two read ports. Hence, Nios V/m processor duplicates the register files so that two different source registers for an instruction are available in a single cycle. After performing ALU operations, the processor core writes the same result to the destination register in both memories.

Table 25. Nios V Processor General-Purpose Register File

Register	ABI	Description	Register	ABI	Description
x0	zero	Hardwired to zero	x16	a6	Function argument 6
x1	ra	Return address	x17	a7	Function argument 7
x2	sp	Stack pointer	x18	s2	Saved register 2
x3	gp	Global pointer	x19	s3	Saved register 3
x4	tp	Thread pointer	x20	s4	Saved register 4
x5	t0	Temporary 0	x21	s5	Saved register 5
x6	t1	Temporary 1	x22	s6	Saved register 6
x7	t2	Temporary 2	x23	s7	Saved register 7
x8	s0/fp	• Saved register 0 • Frame pointer	x24	s8	Saved register 8
x9	s1	Saved register 1	x25	s9	Saved register 9
x10	a0	• Function argument 0 • Return value 0	x26	s10	Saved register 10
x11	a1	• Function argument 1 • Return value 1	x27	s11	Saved register 11
x12	a2	Function argument 2	x28	t3	Temporary 3
x13	a3	Function argument 3	x29	t4	Temporary 4
x14	a4	Function argument 4	x30	t5	Temporary 5
x15	a5	Function argument 5	x31	t6	Temporary 6

3.3.2. Arithmetic Logic Unit

The arithmetic logic unit (ALU) operates on data stored in general-purpose registers. ALU operations take one or two inputs from registers and store the result back into the register.

Table 26. Fundamental Data Operations of the ALU

Category	Description
Arithmetic	Addition and subtraction on signed and unsigned operands.
Relational	Equal, not-equal, greater-than-or-equal, and less-than relational operations (==, != >=, <).
Logical	AND, OR, NOR, and XOR logical operations.
Shift	Logical and arithmetic shift operations.

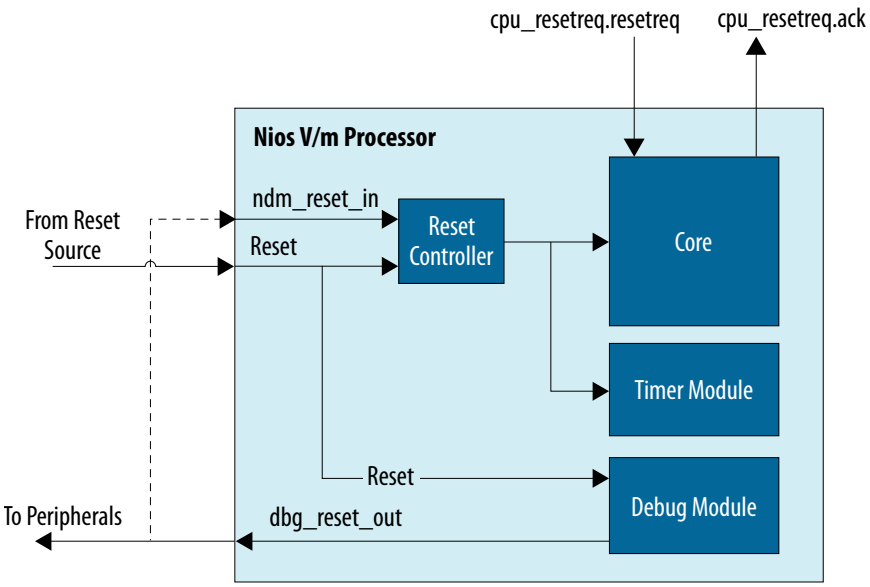
For load and store instructions, the Nios V/m processor uses the ALU to calculate the memory address. For conditional control transfer instructions, Nios V/m processor uses the relational operations in the ALU to determine if the processor takes or leaves the branch.

3.3.3. Reset and Debug Signals

Table 27. Reset and Debug Signals

Interface	Type	Description
`reset`	Reset	A global hardware reset input signal that forces the Nios V processor to reset immediately.
`dbg_reset_out`	Reset	An optional reset output signal which appear after you enable both **Enable Debug** and **Enable Reset from Debug Module** parameters. • This reset output signal is triggered by the JTAG debugger or `niosv-download -r` command. • You can connect this reset output signal to the following input signals: — To the `ndm_reset_in` input to reset the core and the timer module. — To the reset input signal of other components as needed.
`ndm_reset_in`	Reset	An optional reset input signal which appear after you enable both **Enable Debug** and **Enable Reset from Debug Module** parameters. • You can use this signal to trigger reset controller to reset the core and the timer module. • The reset controller synchronizes the reset (hard reset) and `ndm_reset_in` signals.
`cpu_resetreq`	Conduit	An optional local reset ports which appear after you enable **Add Reset Request Interface** parameter. The signal consists of an input `resetreq` signal and an output `ack` signal that trigger the Nios V processor to reset without affecting other components in a Nios V processor system. • You can request a reset to the Nios V processor core by asserting the `resetreq` signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the ack signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

Figure 5. Nios V/m Processor Reset Network



3.3.4. Control and Status Registers

Nios V/m processor's Control and Status Registers (CSR) is both readable and writable. Nios V/m updates the CSR during the E-stage of the pipeline.

During the execution of a Nios V processor application, you may observe the following behaviors:

- CSR write instruction (in E-stage) is stalled due to the pending memory or multicycle instructions (in M-stage).
- CSR write instruction (in E-stage) continues after the pending instructions (in Mstage) are complete.
- If the processor generates an exception during the M-stage, the processor flushes the pending instructions in the pipeline (including the CSR write instruction in the E-stage) and initiates the trap handler to service the exception.

3.3.5. Instruction Cycles

The tables below provides instruction cycles for all types of instructions.

Table 28. Instruction Cycles (Pipelined)

Instructions	Category	Cycles	Pipeline Flush
BEQ, BNE, BLT, BGE, BLTU, BGEU	Branch Taken	4	Yes
	Branch Not Taken	1	-
LB, LH, LW, LBU, LHU	Load (With AXI-4/Avalon® - MM transfer)	More than 1	-
SB, SH, SW	Store (With AXI-4/Avalon® - MM transfer)	More than 1	-

continued...

Instructions	Category	Cycles	Pipeline Flush
ADD, SUB, ADDI	Arithmetic	1	-
SLL, SLLI, SRA, SRAI, SRL, SRLI	Shift	1 to 8	-
SLT, SLTU, SLTI, SLTIU	Compare	1	-
AND, OR, XOR, ANDI, ORI, XORI	Logic Operation	1	-
JAR, JALR	Jump	4	Yes
ECALL, EBREAK	Environment Call and Breakpoint	4	Yes
CSRRW, CSRRS, CSRRC, CSRRWI, CSRRSI, CSRRCI	Control and Status Register	1	-
LUI, AUIPC, FENCE, FENCE.TSO, PAUSE	Others	1	-

Table 29. Instruction Cycles (Non-Pipelined)

Instructions	Category	Cycles
BEQ, BNE, BLT, BGE, BLTU, BGEU	Branch Taken	7
	Branch Not Taken	6
LB, LH, LW, LBU, LHU	Load (With AXI-4/Avalon® -MM transfer)	More than 6
SB, SH, SW	Store (With AXI-4/Avalon® -MM transfer)	More than 6
ADD, SUB, ADDI	Arithmetic	6
SLL, SLLI, SRA, SRAI, SRL, SRLI	Shift	6
SLT, SLTU, SLTI, SLTIU	Compare	6
AND, OR, XOR, ANDI, ORI, XORI	Logic Operation	6
JAR, JALR	Jump	6
ECALL, EBREAK	Environment Call and Breakpoint	6
CSRRW, CSRRS, CSRRC, CSRRWI, CSRRSI, CSRRCI	Control and Status Register	6
LUI, AUIPC, FENCE, FENCE.TSO, PAUSE	Others	1

3.3.6. Trap Controller (CLINT)

In the Nios V processor, trap refers to the transfer of control to a trap handler caused by either an exception or an interrupt.

- Exceptions are synchronous events that originate inside the processor. They are commonly caused by an unusual condition occurring at run time associated with an instruction.
- Interrupts are asynchronous events that originate outside of the processor. They are commonly caused by service requests from system peripherals.

The RISC-V privileged ISA describes the supported trap modes and the control and status registers used and affected by trap handling. Therefore, this document contains information taken from the privileged ISA spec.

Related Information

RISC-V Specification
More information on RISC-V Trap Handling and related CSRs in The RISC-V Instruction Set Manual: Volume II - Privileged Architecture (Version 20240411).

3.3.6.1. Related Control and Status Register

Nios V processor supports only Machine-Level CSRs, which correlates with the fact that the Nios V processor supports Machine-mode (M-mode) only. Machine-Level CSRs serve as the primary privileged level and have low-level access to the machine implementation.

To simplify the following content on Nios V processor trap handling, CSRs or bit fields that require different privilege levels (e.g., User-mode and Supervisor-mode) are removed.

Since the Nios V processor supports M-mode only, the supported interrupts are Machine-level External (MEI), Machine-level Software (MSI), Machine Timer (MTI), and 16 Platform interrupts.

3.3.6.1.1. Machine Status Register (`mstatus`)

The fourth bit field of the status register (mstatus[3]) holds the global interrupt enable bit, MIE. This bit is commonly used to guarantee atomicity regarding interrupt handlers at the current privilege mode.

When a hart is executing, all interrupts are globally,

- Enabled when `MIE=1`, and
- Disabled when `MIE=0`.

An MRET instruction is used to return from a trap in M-mode.

3.3.6.1.2. Machine Trap-Vector Base-Address Register (`mtvec`)

Table 30. Machine Trap-Vector Base Address Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Base[31:2] (WARL)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base[31:2] (WARL)														Mode	

- The BASE field must always be 4-byte boundary aligned.
- The `MODE` field determines the processor's behaviour for asynchronous interrupts.

Table 31. Supported Vector Modes

mtvec	MODE	Description
00	Direct mode	All traps set pc to BASE << 2
01	Vectored mode	All exceptions set pc to BASE << 2 All interrupts set pc to BASE << 2+4*cause
10	Reserved	-
11	Reserved	-

When MODE=Direct ('00'), all traps into M-mode cause the program counter to be set to the address in the BASE field.

When MODE=Vectored ('01'), all synchronous exceptions cause the program counter to be set to the address in the BASE field, whilst asynchronous interrupts cause the program counter to be set to address in the BASE field plus four times the interrupt cause number.

3.3.6.1.3. Machine Interrupt Register (mip and mie)

The Machine Interrupt Pending register (mip) is a 32-bits wide register containing pending interrupts. The Machine Interrupt Enable register (mie) is a 32-bits wide register containing interrupt enable bits.

Each interrupt cause number corresponds to a bit in both mip and mie registers at bit location Exception Code in mcause register. Thus, interrupt n corresponds to bits mip[n] and mie[n].

Table 32. Machine Interrupt Pending register (mip) Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								MTIP	0			MSIP	0		

Table 33. Machine Interrupt Enable register (mie) Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								MTIE	0			MSIE	0		

3.3.6.1.4. Machine Exception Program Counter Register (mepc)

The mepc register is a 32-bits wide read/write register. When a trap is taken into M-mode, mepc holds the virtual address of the instruction that was interrupted or encountered the exception. An MRET instruction jumps to the address stored in mepc.

3.3.6.1.5. Machine Cause Register (mcause)

Table 34. Machine Cause (mcause) Register Fields

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

The mcause register is a 32-bits wide register. When a trap is taken into M-mode, mcause holds a code identifying the event that caused the trap.

The Interrupt field is set when the trap was caused by an interrupt. The Cause field contains the code identifying the last trap event.

Table 35. mcause Value After Trap

Interrupt	Exception Code	Description
1	3	Machine software interrupt
1	7	Machine timer interrupt
1	16-31	16 Platform interrupts, connected through irq[0:15] interface. For example: • Exception Code 16 = irq[0] • Exception Code 31 = irq[15]
1	Others	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal Instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store address misaligned
0	7	Store access fault
0	Others	Reserved

If an instruction raises multiple exceptions, the decreasing priority order indicates which exception is taken and reported in mcause.

Table 36. Exception Priority in Decreasing Priority Order

Priority	Exception Code	Description
1 (Highest)	3	Instruction address breakpoint
2	1	With physical address for instruction: Instruction access fault
3	2	Illegal instruction

continued...

Priority	Exception Code	Description
	0 3 3	Instruction address misaligned Environment break Load/Store address breakpoint
5	4,6	Optionally: Load/Store address misaligned
6	5,7	With physical address for an explicit memory access: Load/Store access fault
7 (Lowest)	4,6	Load/Store address misaligned

3.3.6.1.6. Machine Trap Value Register (`mtval`)

The `mtval` register is a 32-bits wide read/write register. When a trap is taken in M-mode, it is either set to zero or set to an exception specific value.

Table 37. `mtval` value After Exception

Exception Code	Exception Type	mtval value	mepc value
0	Instruction Address Misaligned	Faulting instruction physical address	Faulting instruction physical address
1	Instruction Access Fault		
4	Load Address Misaligned	Faulting load physical address	Interrupted instruction physical address
5	Load Access Fault		
6	Store Address Misaligned	Faulting store physical address	Interrupted instruction physical address
7	Store Access Fault		
3	Breakpoint	Interrupted instruction physical address	Interrupted instruction physical address
2	Illegal Instruction	Faulting instruction	Faulting instruction physical address
-	Others	Set to 0	Interrupted instruction physical address

3.3.6.2. Direct Mode

Direct Mode is a simple and efficient approach for trap controllers to manage interrupts and exceptions in embedded systems. In this method, each trap is assigned a fixed, specified memory address where the trap handling code is stored. When a trap occurs, the trap controller jumps to this address and executes the trap handling code.

The trap handling code:

- Switches to the separate exception stack (if enabled).
- Stores register values onto the stack.
- Determines the type of exception.
- Passes control to the exception dispatcher or the interrupt dispatcher, providing a quick and direct response to the event.

The Direct Mode supports interrupt pre-emption based on privilege mode, i.e., Machine, Supervisor, and User mode. Since the Nios V processor only supports Machine mode (M-mode), Direct Mode does not support interrupt pre-emption based on privilege mode.

3.3.6.2.1. Hardware Implementation

When a trap occurs, the trap controller recognizes the type of interrupt and directly switches the processor to the trap handling code by configuring the pc to the address specified in the BASE field of the `mtvec` register.

Table 38. Address Computation in Direct Nios V Processor Trap Controller

Assumed the BASE field in `mtvec` register is 0x20.

Trap	Address Computation	Description	Results
Exceptions	`pc <= BASE < < 2`	All exceptions and interrupts	`pc <= 0x80`
Interrupts			

3.3.6.2.2. Software Implementation

When a trap (either exception or interrupt) happens, the trap controller directs it through the trap handling code.

Related Information

Nios® V Processor Software Developer Handbook: Trap Handling

3.3.6.3. Vectored Mode

The Vectored Mode is designed to provide low-latency and vectored interrupts for RISC-V systems. Vectored Mode in RISC-V systems efficiently manages traps by directing the processor to the appropriate trap handler using a vector table. The trap controller uses the base address contained in the BASE field of `mtvec` register, to determine the address of the handler based on the `Interrupt` and `Exception Code` field from the mcause register. This method enables quick and precise redirection to the correct handler, improving flexibility and reaction time while managing various traps.

The Vectored Mode supports interrupt pre-emption based on privilege mode, i.e., Machine, Supervisor, and User mode. Since the Nios V processor only supports Machine mode (M-mode), Vectored Mode does not support interrupt pre-emption based on privilege mode.

Note: CLINT-Vectored interrupt mode is not supported in Nios V/m-Non-pipelined processor.

3.3.6.3.1. Hardware Implementation

In Vectored Mode, synchronous exceptions and asynchronous interrupts implement different address calculations. The Nios V processor determines whether the traps are exceptions or interrupts from the Interrupt field of `mcause` register.

For synchronous exceptions, such as illegal instructions, the hardware assigns the pc to the address provided in the BASE field of the `mtvec` register. This base address serves as a fixed entry point for the generic exception handler routine.

When an interrupt occurs, the system computes the next pc address to the base address from the BASE field plus four times the interrupt cause number, which is acquired from `Exception Code` in the `mcause` register. This calculation determines the address of the dedicated interrupt handler in the vector table.

Note: The effective range for jump instructions is limited to within 1 MiB from the current instruction's address.

Table 39. Address Computation in Vectored Nios V Trap Controller

Trap	Address Computation	Description	Exception Code	Results
Exceptions	`pc <= BASE << 2`	All exceptions	Starting from 0 (Instruction address misaligned), up to 19 (Hardware error)	`pc <= 0x80`
Interrupts	`pc <= BASE << 2 + 4*Exception Code`	Machine software interrupt	3	`pc <= 0x8C`
		Machine timer interrupt	7	`pc <= 0x9C`
		Platform interrupt 0	16	`pc <= 0xC0`
		…	…	…
		Platform interrupt 15	32	`pc <= 0x100`

Note: Assume the BASE field in mtvec register is 0x20.

3.3.6.3.2. Software Implementation

When a trap occurs, whether it is an exception or an interrupt, the trap controller directs it to either the exception dispatcher for exceptions or the interrupt service routine for interrupts.

Related Information

Nios® V Processor Software Developer Handbook: Trap Handling

3.3.7. Memory and I/O Organization

You can configure the Nios V/m processor systems. Consequently, the memory and I/O organization varies from system to system. A Nios V/m processor core uses one or more of the following ports to provide access to memory and I/O:

- Instruction manager port: A Memory-Mapped manager port that connects to instruction memory via system interconnect fabric.
- Data manager port: A Memory-Mapped manager port that connects to data memory and peripherals via the system interconnect fabric.

Nios V/m Processor Core Memory Mapped I/O Access: Both data memory and peripherals are mapped into the address space of the data manager port. Nios V/m processor core uses little-endian byte ordering. Words and half-words are stored in memory with the more-significant bytes at higher addresses. The Nios V/m processor core does not specify anything about the existence of memory and peripherals. The quantity, type, and connection of memory and peripherals are system dependent.

3.3.7.1. Instruction and Data Buses

3.3.7.1.1. Instruction Manager Port

Nios V/m processor instruction bus is implemented as a 32-bit AMBA* 4 AXI-Lite manager port as default interface. You can switch it to Avalon memory-mapped interface for smaller logic utilization.

The instruction manager port:

- Performs a single function: it fetches instructions to be executed by the processor.
- Does not perform any write operations.
- Can issue successive read requests before data return from prior requests.
- Can prefetch sequential instructions.
- Always retrieves 32-bit of data. Every instruction fetch returns a full instruction word, regardless of the width of the target memory. The widths of memory in the Nios V/m processor system is not applicable to the programs. Instruction address is always aligned to a 32-bit word boundary.
- Does not require any burst adapter because it is non-bursting.

Table 40. AMBA* 4 AXI-Lite Instruction Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Unused	[31:0]	Output
	awprot	Unused	[2:0]	Output
	awvalid	Unused	1	Output
	awready	Unused	1	Input
Write Data Channel	wdata	Unused	[31:0]	Output
	wstrb	Unused	[3:0]	Output
	wvalid	Unused	1	Output
	wready	Unused	1	Input
Write Response Channel	bresp	Unused	[1:0]	Input
	bvalid	Unused	1	Input
	bready	Unused	1	Output
Read Address Channel	araddr	Instruction Address (Program Counter)	[31:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Instruction address valid	1	Output
	arready	Instruction address ready (from memory)	1	Input
Read Data Channel	rdata	Instruction	[31:0]	Input

continued...

Interface	Signal	Role	Width	Direction
	rresp	Instruction response: Non-zero value denotes instruction access fault exception	[1:0]	Input
	rvalid	Instruction valid	1	Input
	rready	Constant 1	1	Output

Table 41. Avalon Memory-Mapped Instruction Interface Signals

Signal	Role	Width	Direction
readdata	Instruction	[31:0]	Agent → Processor
waitrequest	Force the processor to wait until the agent is ready	1	Agent → Processor
readdatavalid	Instruction is valid	1	Agent → Processor
response	Instruction response: Non-zero value denotes instruction access fault exception	[31:0]	Agent → Processor
address	Instruction Address (Program Counter)	[31:0]	Processor → Agent
read	Asserted to indicate a read transfer.	1	Processor → Agent

3.3.7.1.2. Data Manager Port

Nios V/m processor data bus is implemented as a 32-bit AMBA 4 AXI-Lite manager port as default interface. You can switch it to Avalon memory-mapped interface for smaller logic utilization. The data manager port:

- Performs read data from memory or a peripheral when the processor executes a load instruction.
- Performs write data to memory or a peripheral when the processor executes a store instruction.
- Does not require any burst adapter because it is non-bursting.

axsize signal value indicates the load/store instruction size- byte (LB/SB), halfword (LH/SH) or word (LW/SW). Address on axaddr signal is always aligned to size of the transfer. For store instructions, respective writes strobe bits are asserted to indicate bytes being written.

Nios V/m processor core does not support speculative issue of load/store instruction. Hence, a core can issue only one load or store instruction and waits until the issued instruction is complete.

Table 42. AMBA 4 AXI-Lite Data Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Store address	[31:0]	Output
	awprot	Unused	[2:0]	Output
	awvalid	Store address valid	1	Output
	awready	Store address ready (from memory)	1	Input

continued...

Interface	Signal	Role	Width	Direction
Write Data Channel	wdata	Store data	[31:0]	Output
	wstrb	Byte position in word	[3:0]	Output
	wvalid	Store data valid	1	Output
	wready	Store data ready (from memory)	1	Input
Write Response Channel	bresp	Store response: Non-zero value denotes store access fault exception.	[1:0]	Input
	bvalid	Store response valid	1	Input
	bready	Constant 1	1	Output
Read Address Channel	araddr	Load address	[31:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Load address valid	1	Output
	arready	Load address ready (from subordinates)	1	Input
Read Data Channel	rdata	Load data	[31:0]	Input
	rresp	Load response: Non-zero value denotes load access fault exception	[1:0]	Input
	rvalid	Load data valid	1	Input
	rready	Constant 1	1	Output

Table 43. Avalon Memory-Mapped Data Interface Signals

Signal	Role	Width	Direction
readdata	Load data	[31:0]	Agent → Processor
waitrequest	Force the processor to wait until the agent is ready	1	Agent → Processor
readdatavalid	Load data is valid	1	Agent → Processor
response	Load/Store response: Non-zero value denotes access fault exception	[31:0]	Agent → Processor
address	Load/Store address	[31:0]	Processor → Agent
read	Asserted to indicate a read transfer	1	Processor → Agent
write	Asserted to indicate a write transfer	1	Processor → Agent
writedata	Store data	[31:0]	Processor → Agent
byteenable	Indicates the load/store instruction size • Byte (LB/SB), • Halfword (LH/SH), or • Word (LW/SW)	[3:0]	Processor → Agent
writeresponsevalid	Asserted to indicate a valid write response	1	Agent → Processor

3.3.7.1.3. Choosing a Suitable Interface

The Nios V/m processor core offers an Avalon memory-mapped and Advanced eXtensible Interface (AXI) interface for both the instruction manager port and the data manager port. You can choose one of the interfaces in Platform Designer based on your design.

For example, if most IPs in Platform Designer use Avalon memory-mapped interface, you can choose the Avalon memory-mapped interface in the Nios V/m processor core as the bridge interface. Similarly, if most IPs in Platform Designer use AXI Interface, you can choose the AXI Interface in the Nios V/m processor core as the bridge interface.

The benefits of using the appropriate interface are as follows:

- Reduces the amount of bridging logic
- Reduces overhead bandwidth
- Provides better performance

3.3.7.2. Address Map

The address map for memories and peripherals in a Nios V/m processor system is design dependent. The following addresses are part of the processor:

1. Reset Address
2. Debug Exception Address
3. Exception Address
4. Timer and Software Interrupt Address

You can specify the Reset Address and Debug Exception Address in Platform Designer during system configuration. You can modify the Exception Address stored in the `mvtec` register. `mvtime` and `mtimecmp` register controls the timer interrupt. The `msip` register bit controls the software interrupt.

3.3.8. RISC-V based Debug Module

The Nios V/m processor architecture supports a RISC-V based 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 debug module and provide facilities, such as the following features:

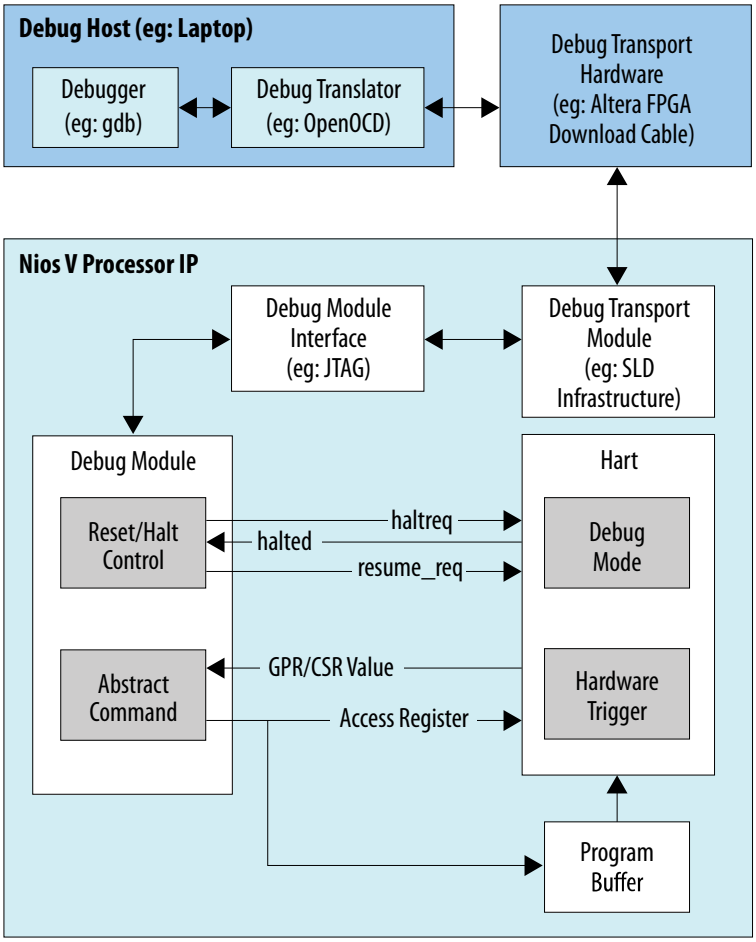
- Reset Nios V processor core and timer module
- Download programs to memory
- Start and stop execution
- Set software breakpoints and watchpoints
- Analyze registers and memory

You can interact with the Debug Host (e.g., laptop), which is running a debugger (e.g., gdb). The debugger communicates with a Debug Translator (e.g., OpenOCD) to communicate with Debug Transport Hardware (e.g., Altera FPGA Download Cable). The Debug Transport Hardware connects the Debug Host to the Nios V processor's Debug Transport Module (DTM). The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

Every Nios V processor has a single hart, which is controlled by a single DM. DMs provide reset/halt control of the Nios V processor. Abstract commands provide access to GPRs. Additional registers are accessible through abstract commands or the Program Buffer. In addition to register access, the Program Buffer also allows the debugger to execute arbitrary instructions and access memory.

The Nios V processor implements a Trigger Module. When trigger conditions are met, the processor halts and informs the debug module that it has halted.

Figure 6. Nios V Processor Debug System Overview



3.3.8.1. Debug Module

The Debug Module supports the following operations:

1. Provide information about the processor's implementation to the debugger.
2. Allow users to halt and resume the processor.
3. Provide halt status on the processor.
4. Provide abstract read and write access to the halted processor's GPRs through abstract command.

5. Provide `ndm_reset_in` and `dbg_reset_out` signals that allow debugging from the reset vector after reset.
6. Provide abstract access to non-GPR hart registers through abstract command.
7. Provide a Program Buffer to force the hart to execute arbitrary instructions through abstract commands.

The Nios V processor supports only one hart. Therefore:

- You cannot apply hart selection.
- The bit fields that differentiate between "all harts" and "any harts" are treated as the same.
- It simplifies related bit fields(such as `dmstatus.allhavereset` and `dmstatus.anyhavereset` to `dmstatus.*havereset` throughout the document.

3.3.8.1.1. Reset/Halt/Run Control

In the Nios V processor, the Debug Module tracks 3 conceptual bits of state:

- halt request
- resume ack
- hart reset

In the `dmstatus` register, the debugger can observe the state of resume ack in `dmstatus.*resumeack`, and the state of halted, running, and havereset signals in `dmstatus.*halted`, `dmstatus.*running`, and `dmstatus.*havereset`.

Reset processor

Debug Module controls a global reset signal, `dmcontrol.ndmreset` (non-debug module reset), which can reset, or hold in reset, every component in the platform, except for the Debug Module and Debug Transport Modules. The debugger can reset the processor by asserting `dmcontrol.ndmreset`, through the `ndm_reset_in` and `dbg_reset_out` signals. Refer to Reset and Debug Signals about how to connect the reset signals. Next, check the reset status using `dmstatus.*havereset` bit field.

Halt processor

When the processor comes out of reset and `dmcontrol.haltreq` are set, the hart immediately halt and enter Debug Mode (halted state). Next, check the halted status using `dmstatus.*halted` bit field.

Resume processor

To resume the processor's execution, debugger writes 1 to `dmcontrol.resumereq`. Next, check the resume status using `dmstatus.*resumeack` bit field. After that, `dmstatus.*running` is set to 1 indicating the processor is running in normal execution.

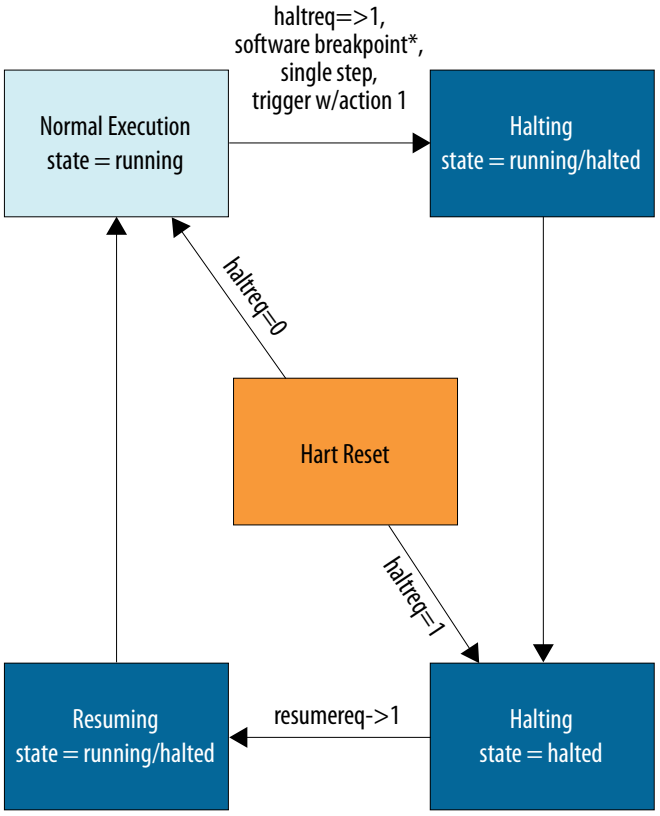
3.3.8.1.2. Hart States

The processor is under one of the following four states at any given time:

- Non-existent (Probing other harts which does not exist)
- Unavailable (Processor being reset)
- Running (Processor executing normally outside of debug)
- Halted (Processor in Debug Mode)

They are reflected in the `dmstatus` register, as `dmstatus.*nonexistent`, `dmstatus.*unavail`, `dmstatus.*running` and `dmstatus.*halted` respectively.

Figure 7. Relationship between Hart DM States



3.3.8.1.3. Debug Module Register

The Debug Module registers are accessible over the DMI bus only. Each register has a fixed address and developed based on the RISC-V Debug Specification 0.13.2, with JTAG DTM. The Debugger can determine the register implementation status by writing to or reading from the Debug Module registers. Unimplemented registers return 0 when read.

Table 44. Debug Module Register

Address	Name	Description
0x04	Abstract Data 0 (`data0`)	Refer to Abstract Commands.
0x05	Abstract Data 1 (`data1`)	Refer to Abstract Commands.
0x10	Debug Module Control (`dmcontrol`)	• There is a single hart selected. • `hartsel` is 0.
0x11	Debug Module Status (`dmstatus`)	• Supports implicit ebreak instruction after Program Buffer execution. Authentication is not implemented. `authenticated` and `authbusy` hardwired to 1. • `confstrptr` is not relevant to configuration string. • Halt-on-reset functionality is not supported. • Version = 3
0x12	Hart Info (`hartinfo`)	• No `dscratch` registers available during Program Buffer execution. • `data0` holds arg0/return value • `data1` holds arg1 • No CSRs dedicated to shadowing the `data` registers.
0x14	Hart Array Window Select (`hawindowsel`)	Unimplemented register. Nios V processor contains a single hart only.
0x15	Hart Array Window (`hawindow`)	Unimplemented register. Nios V processor contains a single hart only.
0x16	Abstract Control and Status (abstractcs)	• Eight (8) Program Buffer, in 32-bits words. • Two (2) `data` registers Refer to Abstract Commands.
0x17	Abstract Command (command)	Refer to Abstract Commands.
0x18	Abstract Command Autoexec (abstractauto)	Unimplemented register.
0x19 ~ 0x1c	Configuration String Pointers (confstrptr)	Unimplemented registers.
0x1d	Next Debug Module (nextdm)	Unimplemented register. There is only a single DM.
0x20 ~ 0x27	Program Buffer 0 to 7	Refer to Program Buffer.
0x30	Authentication Data	0x0
0x37 ~ 0x3f	System Bus Registers	Unimplemented registers. System Bus is not supported.
0x40	Halt Summary 0	LSB reflects halt status of the Nios V processor.

3.3.8.2. Abstract Commands

Nios V processor DM supports Access Register Command only (`command.cmdtype` = 0). You can determine that only Access Register abstract commands is supported by attempting them and then check the abstractcs.cmderr. Other abstract commands are not supported (`abstractcs.cmderr` = 2).

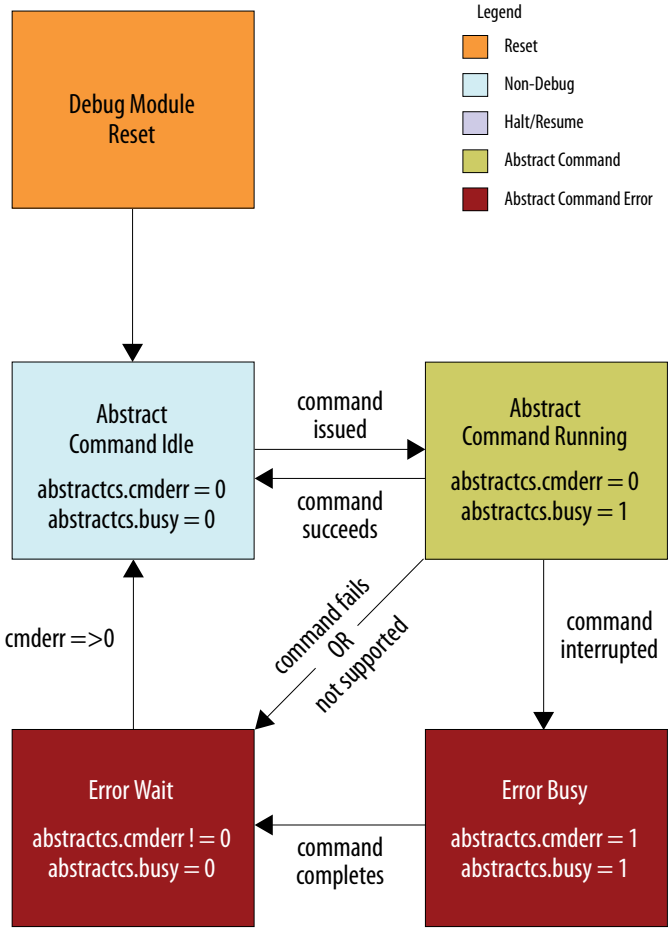
Debuggers execute abstract commands by writing them to command and check its completion by reading `abstractcs.busy`. If the debugger starts a new command while abstractcs.busy is set, `abstractcs.cmderr` becomes 1 (busy), the currently

executing command continues without interruption, but any error generated from its execution is lost. After completion, `abstractcs.cmderr` indicates whether the command was successful. Commands may fail because the processor is not halted, not running, unavailable, or encounter an error during execution.

If the command takes arguments, the debugger must write them to the data registers before writing to `command`. If a command returns results, the results are placed in the `data` registers before `abstractcs.busy` is cleared.

Note: The Nios V processor does not support abstract commands when hardware thread is not halted.

Figure 8. Behaviour of Abstract Commands



3.3.8.2.1. Access Register Command

The Access Register command allows read-write access to the processor registers including GPRs, CSRs, FP registers and Program Counter (dpc in Debug Mode). The Access Register also allows program execution from program buffer. The debugger executes Access Register commands by writing into `command` register using the Access Register command encoding.

Sequence to execute Access Register Command:

1. If `command.write` is 0 and `command.transfer` is 1, then read data from the register specified by command.regno into the data0.
2. If `command.write` is 1 and `command.transfer` is 1, then write data from the data0 into the register specified by `command.regno`.
3. If `command.aarpostincrement` is 1, increment `command.regno`.
4. If `command.postexec` is 1, execute the Program Buffer.

Figure 9. Accessing Access Register Command

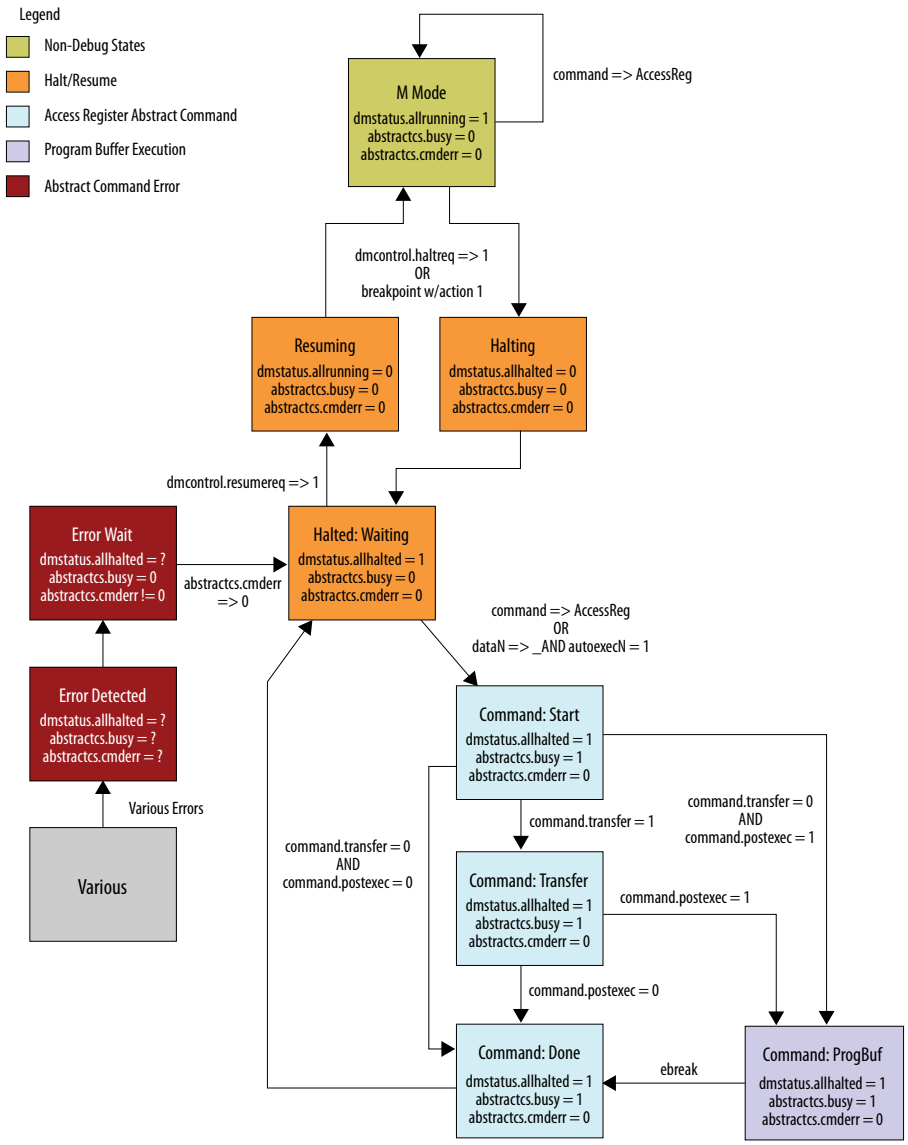


Table 45. Access Register Command Encoding

Bit Field							
31 - 24	23	22 - 20	19	18	17	16	15-0
cmdtype=0	0	aarsize=2	aarpostincrement	postexec	transfer	write	regno

Table 46. Field Description

Field	Description
cmdtype	0 indicates Access Register command.
aarsize	2 indicates 32-bit register access. Others are not applicable in 32-bit Nios V processor.
aarpostincrement	0: No effect 1: Increment regno after successful register access
postexec	0: No effect 1: Execute program in program buffer
transfer	Acts in conjunction with write field. 0: Ignore value in write field 1: Execute operation specified by write field.
write	0: Copy data from register (Read) 1: Copy data to register (Write)
regno	Register address to be accessed • 0x0 ~ 0xfff : CSRs (Refer to the topic Control and Status Registers (CSR) Mapping (CLINT) for available CSRs). • 0x1000 ~ 0x101f : GPRs • 0x1020 ~ 0x103f : Floating Point Register

If the Access Register command fails, the DM stores root causes in the `abstractcs.cmderr` field. The `abstractcs.cmderr` field represents the current state of abstract command being executed. If the `abstractcs.cmderr` field is non-zero, it ignores the writes to the command register. To clear the `abstractcs.cmderr` field, write 1 for every bit in the field.

Table 47. cmderr Description

cmderr	Description
0	No error.
1	DM is busy.
2	The command is not supported.
3	An exception occurred when executing the command.
4	The processor is not halted.
5	Experiencing bus error.
7	Others

For more information about Access Register command, refer to The RISC-V Debug Specification.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

3.3.8.2.2. Using Access Register Command

Read `s0` using abstract command:

Op	Address	Value	Comment
Write	`command`	`aarsize=2, transfer, regno= 0x1008`	Read s0
Read	`data0`	—	Returns value that was in s0

Write `mstatus` using abstract command:

Op	Address	Value	Comment
Write	`data0`	new value	—
Read	`command`	`aarsize=2, transfer, regno= 0x300`	Write `mstatus`

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

3.3.8.3. Program Buffer

Debugger can write small programs into Program Buffer to execute arbitrary instruction, then use Access Register command to initiate the execution (setting `command.postexec`).

Note that the Nios V processor Debug Module supports implied ebreak at the non-existent word immediately after the Program Buffer. This saves the debugger from having to write the ebreak itself, and allows the sequence of instructions to be one word smaller.

3.3.8.3.1. Using Program Buffer

Table 48. Write `mstatus` using Program Buffer

Op	Address	Value	Comment
Write	`progbuf0`	`csrw s0, MSTATUS`	—
Write	`progbuf1`	`ebreak`	—
Write	`data0`	`new value`	—
Write	`command`	`aarsize=2, postexec, transfer, write, regno= 0x1008`	Write s0, then excecute program buffer.

Table 49. Read `f1` using Program Buffer

Op	Address	Value	Comment
Write	`progbuf0`	`fmv.x.s s0, f1`	—
Write	`progbuf1`	`ebreak`	—

continued...

Op	Address	Value	Comment
Write	command	postexec	Execute program buffer
Write	command	transfer, regno=0x1008	read s0
Read	data0	—	Returns the value that was in f1

Table 50. Read a Word from Memory using Program Buffer

Op	Address	Value	Comment
Write	progbuf0	lw S0, 0(s0)	—
Write	progbuf1	ebreak	—
Write	data0	address	—
Write	command	write, postexec, regno=0x1008	Write s0, then execute program buffer
Write	command	regno=0x1008	Read s0
Read	data0	—	Value read from memory

Table 51. Write a Word to Memory Using Program Buffer

Op	Address	Value	Comment
Write	progbuf0	sw s1, 0(s0)	—
Write	progbuf1	ebreak	
Write	data0	address	—
Write	command	write, regno=0x1008	Write s0
Write	data0	value	—
Write	command	write, postexec, regno=0x1009	Write s1, then execute program buffer.

3.3.8.4. Debug Mode

3.3.8.4.1. Enter Debug Mode

You can enter the Debug Mode, as specified in the RISC-V architecture specification, in the following ways:

1. Halt from Debug Module
2. Software breakpoints
3. Trigger or Hardware breakpoint

Upon entering Debug Mode, Nios V processor completes the instruction in W-stage. By the order of priority, instruction in M-stage, E-stage, D-stage or F-stage takes the interrupt.

- When there is a valid instruction, Program Counter writes to the Debug Program Counter, `.dpc`.
- If the instruction in M-stage is not valid, then instruction in E-stage takes the interrupt and so on and so forth.
- If there is no valid instruction in the pipeline, the Program Counter for the next instruction writes to the Debug Program Counter, `.dpc`.

Note: For branches, the next Program Counter depends on whether a branch was taken or not taken, and whether the branch prediction (if any) was correct.

3.3.8.4.2. Exit Debug Mode

Executing the dret instruction in Debug Mode changes pc to the value stored in dpc. The processor then enters Machine Mode and exits Debug Mode.

3.3.8.4.3. Core Debug Registers

These registers are only accessible from Debug Mode, using the RISC-V csr opcodes or abstract debug commands.

Table 52. Core Debug Registers

Name	Name	Description
dcsr	Debug Control and Status	• ebreak instructions in M-mode enter Debug Mode. • Interrupts are disabled during single stepping. • Increment counters and timers as usual. • MPRV in mstatus is ignored in Debug Mode.
dpc	Debug PC	—
dscratch0	Debug Scratch Register 0	—

3.3.8.5. Hardware Trigger Module

The Nios V processor core supports one address or data match trigger. The trigger registers are accessible using RISC-V csr opcodes or abstract debug commands. The firing of the trigger can either enter the Debug Mode or raise a breakpoint exception, which depends on the trigger registers.

Without applying software breakpoint, this makes hardware triggers invaluable when debugging code from ROM.

Note: The trigger is disabled in the following situations:

- The Nios V processor is in debug mode.
- The Nios V processor is in machine mode, while `mcontrol.m` or `mcontrol.type` is 0.

3.3.8.5.1. Trigger Registers

Table 53. Trigger Registers Implemented in Nios V Processor

Name	Registers	Description
`tselect`	Trigger Select	Nios V processor supports one trigger, therefore the processor selects Trigger 0 at default (value set at 0).
`tdata1`	Trigger Data 1	The value of type field is 2 to represent Trigger 0 as an address or data match trigger. Write behavior to tdata registers depends on the dmode filed. The remaining bits acts as `mcontrol`.
`mcontrol`	Match Control	Controls address and data trigger implementation according to `action`, `m`, `execute`, `store`, and `load` fields.
`tdata2`	Trigger Data 2	Holds trigger-spesific data (virtual address, instruction opcode, data stored or loaded).
`tinfo`	Trigger Info	The value of info field is 4 at default to signify `tdata1.type` is 2. This implies selected trigger is and address or data match trigger.

3.3.8.5.2. Type of Address/Data Match Triggers

Base on the bit field setting in `mcontrol` register, Nios V processor can implement different trigger types after the selected trigger matches the tdata2 register.

You can configure the hardware triggers to fire (through `mcontrol` register), when matches:

- Instruction address
- Instruction opcode
- Store address
- Store data
- Load address
- Load data

Table 54. Address or Data Match Trigger Type

Trigger	`mcontrol` bit fields			
	`select`	`execute`	`store`	`load`
Instruction Address	0	1	0	0
Instruction Opcode	1	1	0	0
Store Address	0	0	1	0
Store Data	1	0	1	0
Load Address	0	0	0	1
Load Data	1	0	0	1

Based on the selected trigger, the trigger firing time and the behaviour of the `mepc` register can varies.

Table 55. Nios V Processor Triggers Condition and Firing Time

Triggers	Condition	Firing Time	Exception Program Counter
Instruction Address Trigger	Program Counter matches `tdata2`	Before executing the instruction	The processor sets the Machine Exception Program Counter (`mepc`) to the instruction address (PC).
Instruction Opcode Trigger	Instruction opcode matches `tdata2`	Before executing the instruction	
Store Address Trigger	Store address matches `tdata2`	After executing the store instruction	The processor sets the `mepc` to the next instruction address (PC + 4).
Store Data Trigger	Store data matches `tdata2`	After executing the store instruction	
Load Address Trigger	Load address matches `tdata2`	After executing the load instruction	
Load Data Trigger	Load data matches `tdata2`	After executing the load instruction	

3.3.8.5.3. Using Triggers

A debugger can use hardware triggers to halt a hart (or enter Debug Mode) when a certain event occurs. In Nios V processor, the supported trigger type is address/data match trigger. Thus, the associated events are related to address and data trigger implementation only. When a debugger wants to set a trigger, the debugger writes the desired configuration, and then reads back to see if that configuration is supported.

Table 56. Using Triggers Example 1

Enter Debug Mode just before the instruction at 0x8001234 is executed. Use as an instruction breakpoint in ROM.

Name	Address	Description
`tdata1`	0x105c	action=1, match=0, m=1, s=1, u=1, execute=1
`tdata2`	0x80001234	address

Table 57. Using Triggers Example 2

Enter Debug Mode right after the value at 0x80007f80 is read.

Name	Address	Description
`tdata1`	0x4159	timing-1, action=1, match=0, m=1, s=1, u=1, load=1
`tdata2`	0x80007f80	address

Table 58. Using Triggers Example 3

Enter Debug Mode right before a write to an address between 0x81230007c80 and 0x80007cef (inclusive).

Name	Address	Description
`tdata1 0`	0x195a	action=1, chain=1, match=2, m=1, s=1, u=1, store=1
`tdata2 0`	0x80007c80	start address (inclusive)
`tadata1 1`	0x11da	action=1, match=3, m=1, s=1, u=1, store=1
`tdata2 1`	0x80007cf0	end address (exclusive)

Table 59. Using Triggers Example 4

Enter Debug Mode right before a write to an address between 0x8123000 and 0x8123ffff.

Name	Address	Description
tdata1	0x10da	action=1. match=1, m=1, s=1, u=1. store=1
tdata2	0x81237fff	16 bits to match exactly, then 0, then all ones

Table 60. Using Triggers Example 5

Enter Debug Mode right after a read from an address between 0x86753090 and 0x8675309f or between 0x96753090 and 0x96753309f (inclusive).

Name	Address	Description
tdata1 0	0x41a59	timing=1, action=1, chain=1, match=4, m=1, s=1. u=1, load=1
tdata2 0	0xfff03090	Mask for low half, then match for low half
tadata1 1	0x412d9	timing=1, action=1, match=5, m=1, s=1, u=1, load=1
tdata2 1	0xefff8675	Mask for high half, then match for high half

3.3.8.6. JTAG Debug Transport Module

The JTAG DTM provides access to the DM over JTAG, developed based on RISC-V Debug Specification 0.13.2, with JTAG DTM.

3.3.8.6.1. JTAG DTM Registers

When the JTAG TAP is reset, IR is 00001 at default (selecting the IDCODE instruction).

Table 61. JTAG DM Register

Address	Name	Description
0x00	BYPASS	JTAG recommends this encoding
0x01	IDCODE	JTAG recommends this encoding
0x10	DTM Control and Status (dtmcs)	Refer to .
0x11	Debug Module Interface Access (dmi)	Refer to *RISC-V Debug Specification 0.13.2, with JTAG DTM*.
0x1f	BYPASS	JTAG requires this encoding

3.3.8.6.2. Accessing Debug Module Interface

Altera recommends you to utilize the OpenOCD executables along with the RISC-V Debug Configuration, Authentication, and DMI commands to access the Debug Module. They offer abstraction to the JTAG DTM, which reduces complexity and improve efficiency during Debug Module access.

If you are developing a custom Debug Translator, please refer to subchapter *B.1 Debug Module Interface Access* in *RISC-V Debug Specification 0.13.2, with JTAG DTM*.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

3.3.8.7. Nios V Processor Implementation

The Nios V processor implements the Execution-Based. This implementation only implements the Access Register abstract command for GPRs on a halted hart, and relies on the Program Buffer for all other operations. It uses the hart's existing pipeline and ability to execute from arbitrary memory locations to avoid modifications to a hart's datapath.

For more information about Execution Based implementation, refer to *RISC-V Debug Specification 0.13.2, with JTAG DTM* in Appendix *Execution Based*.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

3.3.9. Error Correction Code (ECC)

The Nios V/m processor core has the option to enable error detection and ECC status reporting for the RAM block, that is the Register file. Each RAM block has its own source ID. When an ECC event occurs, the processor transmits the source ID and ECC status to the ECC interface.

- If the ECC event is a correctable error, the processor continues to operate after correcting the error. The correction made is not written back to its memory source.
- If the ECC event is an un-correctable error, the processor halts its current progress and stalls. You need to reset either the processor core alone or the entire system.

Note: To reset the processor core alone, you need to apply the **Reset Request Interface** to safely reset the Nios V processor (cleared of any outstanding operations). To reset the entire system, you can use the hard reset interface instead.

The ECC interface allows external logic to monitor ECC errors from the Nios V/m processor. The interface is a conduit, made up of the following output signals.

- `cpu_ecc_status` : Indicates the error status
- `cpu_ecc_source` : Indicates the error source.

Table 62. `cpu_ecc_status`

2-bits Encoding	Description	Effects on Software
2'b00	No ECC event	None
2'b01	Reserved	Not Applicable
2'b10	Correctable single bit ECC error	None
2'b11	Un-correctable ECC error	Likely fatal and halts the processor

Table 63. cpu_ecc_source

4-bits Encoding	ECC Source	Available
4'b0000	No ECC event	Always
4'b0001	General Purpose Register (GPR)	Always
4'b0010 ~ 4'b1110	Other RAM Blocks	Not Available
4'b1111	Reserved	Not Applicable

Related Information

Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide

3.4. Programming Model

3.4.1. Privilege Levels

The privilege levels in Nios V/m processor are designed based on the RISC-V architecture specification. The supported privilege levels available are Machine Mode (M-mode) and Debug Mode (D-mode).

Related Information

RISC-V® Specifications

3.4.1.1. Machine Mode (M-mode)

The default operating mode is Machine mode (M-mode). The core boots up into M-mode after power-on reset. M-mode provides low-level access to the machine implementation and all CSRs.

3.4.1.2. Debug Mode (D-mode)

The Debug mode (D-mode) is an additional privilege level to support off-chip debugging and manufacturing test.

The core can enter D-mode using any of the following methods:

- After reset when bit is set in debug CSR.
- Using debug interrupt.
- Using EBREAK instruction when bit is set in debug CSR.
- Using single step.

3.4.2. Control and Status Registers (CSR) Mapping

Control and status registers report the status and change the behavior of the processor. Since the processor core only supports M-mode and D-mode, Nios V/m processor implements the CSRs supported by these two modes.

Table 64. Control and Status Registers List

Number	Privilege	Name	Description
Machine Trap Setup			
0x300	MRW	mstatus	Machine status register. Refer to Machine Status Register Fields table.
0x301	MRW	misa	ISA and extensions. Refer to Machine ISA Register Fields table.
0x304	MRW	mie	Machine interrupt-enable register. Refer to Machine Interrupt-Enable Register Fields table.
0x305	MRW	mtvec	Machine trap-handler base address. Refer to Machine Trap-Handler Base Address Register Fields table.
Machine Trap Handling			
0x341	MRW	mepc	Machine exception program counter. Refer to Machine Exception Program Counter Register Fields table.
0x342	MRW	mcause	Machine trap cause. Refer to Machine Trap Cause Register Fields table.
0x343	MRW	mtval	Machine bad address or instruction. Refer to Machine Trap Value Register Fields table.
0x344	MRW	mip	Machine interrupt pending. Refer to Machine Interrupt-Pending Register Fields table.
Trigger Registers			
0x7A0	MRW	tselect	Trigger select. Refer to Trigger Select Register Fields table.
0x7A1	MRW	tdata1 (mcontrol)	Trigger data 1 (Match Control). Refer to Trigger Data 1 (Match Control) Register Fields table.
0x7A2	MRW	tdata2	Trigger data 2. Refer to Trigger Data 2 Register Fields table.
0x7A4	MRO	tinfo	Trigger info. Refer to Trigger Info Register Fields table.
Debug Mode Registers			
0x7B0	DRW	dcsr	Debug control and status register. Refer to Debug Control and Status Register Fields table.
0x7B1	DRW	dpc	Debug Program Counter. Refer to Debug Program Counter Register Fields table.
Machine Information Register			
0xF11	MRO	mvendorid	Vendor ID. Refer to Vendor ID Register Fields table.
0xF12	MRO	marchid	Architecture ID. Refer to Architecture ID Register Fields table.
0xF13	MRO	mimpid	Implementation ID. Refer to Implementation ID Register Fields table.
0xF14	MRO	mhartid	Hardware thread ID. Refer to Hardware Thread ID Register Fields table.

Table 65. Memory-mapped Registers

Memory Mapped Address Number	Privilege	Name	Description
ALT_CPU_MTIME_OFFSET	MRW	mtime	Machine timer. Refer to Machine Timer Register Fields table.

continued...

Memory Mapped Address Number	Privilege	Name	Description
(Refer to the system.h.)			
ALT_CPU_MTIME_OFFSET + 0x8 (Refer to the system.h.)	MRW	mtimecmp	Machine timer compare register. Refer to Machine Timer Compare Register Fields table.
ALT_CPU_MTIME_OFFSET + 0x10 (Refer to the system.h.)	MRW	msip	Custom memory-mapped register to assert software interrupt into Nios V processor. Refer to Machine Software Interrupt Register Fields table.

Related Information

The RISC-V® Instruction Set Manual Volume II: Privileged Architecture
More information about M-mode CSR and their respective fields.

3.4.2.1. Control and Status Register Field

The value in the each CSR registers determines the state of the Nios V/m processor. The field descriptions are based on the RISC-V specification.

All CSRs can be accessed by using csrr* instruction in Machine mode except for the Debug mode registers (0x7B0 and 0x7B1) which can only be accessed through Debug mode.

Table 66. Machine Status Register Fields

The mstatus register is a 32-bit read-write register that keeps track of and controls the hart's current operating state.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SD	0								0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0		MPP[1:0]		0		0	MPIE	0	0	0	MIE	0	0	0

The following bitfields are read-only 0 because:

- Bit 22 (TSR = 0): S-mode is not supported
- Bit 21 (TW = 0): There are no modes less privileged than M-mode.
- Bit 20 (TVM = 0): S-mode is not supported
- Bit 19 (MXR = 0): S-mode is not supported
- Bit 18 (SUM = 0): S-mode and U-mode are not supported
- Bit 17 (MPRV = 0): U-mode is not supported
- Bit 16 and Bit 15 (XS[1:0] = 0): S-mode is not supported
- Bit 14 and Bit 13 (FS[1:0] = 0): "F" extension for Single-Precision Floating-Point is not supported
- Bit 10 and Bit 9 (VS[1:0] = 0): S-mode is not supported
- Bit 8 (SPP = 0): S-mode is not supported

- Bit 6 (UBE = 0): U-mode is not supported
- Bit 5 (SPIE = 0): S-mode is not supported
- Bit 1 (SIE = 0): S-mode is not supported

Table 67. Machine ISA Register Fields

The `misa` CSR is a read-write register reporting the ISA supported by the hart.

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

Table 68. Machine Interrupt-Enable Register Fields

The `mie` register is a 32-bit read-write register that contains interrupt enable bits. This core does not support supervisor or user modes. Thus, any interrupt related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MEIE	0	0	0	MTIE	0	0	0	MSIE	0	0	0

Table 69. Machine Trap-Handler Base Address Register Fields

The `mtvec` register is a 32-bit read/write register that holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE).

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

Table 70. Machine Exception Program Counter Register Fields

The `mepc` register is a 32-bit read-write register that holds the addresses of the instruction that was interrupted or that encountered the exception when a trap is taken into M-mode.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mepc														

Table 71. Machine Trap Cause Register Fields

The `mcause` register is a 32-bit read-write register that hold the code indicating the event that caused the trap when a trap is taken into M-mode.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Interrupt	Exception code [30:16]														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Exception code [15:0]															

Table 72. Machine Trap Value Register Fields

The `mtval` register is a 32-bit read-write register that is written with exception-specific information to assist software in handling the trap when a trap is taken into M-mode.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mtval														

Table 73. Machine Interrupt-Pending Register Fields

The `mip` register is a 32-bit read/write register containing information on pending interrupts. This core does not support supervisor or user modes. Thus, any interrupt related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MEIP	0	0	0	MTIP	0	0	0	MSIP	0	0	0

Table 74. Trigger Select Register Fields

The `tselect` register is a 32-bit read/write register that selects the current trigger is accessible by other trigger register.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
`tselect`														

Table 75. Trigger Data 1 (Match Control) Register Fields

The `tdata1 (mcontrol)` register is a 32-bit read/write register containing information on the trigger type, tdata registers accessibility, and trigger implementation. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
type=2				dmode	maskmax						hit	select	timing	sizelo	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
action				chain	match				m	0	0	0	execute	store	load

Table 76. Trigger Data 2 Register Fields

The `tdata2` register is a 32-bit read/write register containing the trigger-specific data.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
tdata2														

Table 77. Trigger Info Register Fields

The `tinfo` register is a 32-bit read-only register containing information on each possible tdata1.type.

Bit Field													
31	30	29	…	18	17	16	15	14	13	…	2	1	0
0							info						

Table 78. Debug Control and Status Register Fields

The `dcsr` CSR is a 32-bit read-write register containing information and status during D-mode. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
debugver				0											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ebreakm	0	0	0	stepie	stopcount	stoptime	cause			0	mprven	nmip	step	prv	

Table 79. Debug Program Counter Register Fields

Upon entry to D-mode, `dpc` CSR is updated with the virtual address of the next instruction to be executed.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
dpc														

Table 80. Vendor ID Register Fields

The `mvendorid` CSR is a 32-bit read-only register that provides the JEDEC manufacturer ID of the provider of the core.

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

Table 81. Architecture ID Register Fields

The `marchid` CSR is a 32-bit read-only register encoding the base microarchitecture of the hart.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
Architecture ID														

Table 82. Implementation ID Register Fields

The `mimpid` CSR provides a unique encoding of the version of the processor implementation.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Implementation														

Table 83. Hardware Thread ID Register Fields

The `mhartid` CSR is a 32-bit read-only register that contains the integer ID of the hardware thread running the code

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Hart ID														

3.4.2.2. Memory Mapped Registers Fields

A RISC-V processor applies LOAD or STORE instruction to access the memory-mapped registers. In the context of Nios V processor HAL API, the LOAD or STORE instructions are referred as IORD or IOWR instruction respective.

The Nios V processor HAL API provides macro to read or write these registers in:

- `bsp\HAL\inc\intel_niosv.h` for machine timer-related registers. The field descriptions are based on the RISC-V specification.
- `bsp\HAL\inc\sys\msw_interrupt.h` for the custom Machine Software Interrupt register.

Table 84. Machine Timer Register Field

The `mtime` register is a 64-bit read-wite register that keeps track of the number of clock cycles.

Bit Field														
63	62	61	60	59	58	57	...	6	5	4	3	2	1	0
`mtime`														

Table 85. Machine Timer Compare Register Field

The `mtimecmp` register is a 64-bit read-wite register that holds a reference value. A machine timer interrupt becomes pending whenever mtime contains a value greater than or equal to mtimecmp,

Bit Field														
63	62	61	60	59	58	57	...	6	5	4	3	2	1	0
`mtimecmp`														

Table 86. Custom Machine Software Interrupt Register Field

The `msip` register is a 32-bit read-wite register that asserts the machine software interrupt. The machine software interrupt becomes pending when msip (bit field 0) is 1, and vice versa.

Bit Field													
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Hardwired to 0														`msip`

3.5. Core Implementation

3.5.1. Instruction Set Reference

The Nios V/m processor is based on the RV32IZicsr specification. RV32IZicsr includes the following instructions sets:

- RV32I base integer instruction set
- Control and status register instructions ("Zicsr" extension).

There are 6 types of instruction formats. They are R-type, I-type, S-type, B-type, U-type, and J-type.

Table 87. Instruction Formats (R-type)

Bit Field (R-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
funct7							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd					opcode						

Table 88. Instruction Formats (I-type)

Bit Field (I-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:0]												rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd					opcode						

Table 89. Instruction Formats (S-type)

Bit Field (S-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:5]							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:0]					opcode						

Table 90. Instruction Formats (B-type)

Bit Field (B-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[12]	imm[10:5]						rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:1]			imm[11]		opcode						

Table 91. Instruction Formats (U-type)

Bit Field (U-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[31:16]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Table 92. Instruction Formats (J-type)

Bit Field (J-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[20]	imm[10:1]										imm[11]	imm[19:16]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Related Information

The RISC-V® Instruction Set Manual Volume I: Unprivileged ISA
More information about the the `fence.i` instruction in RV32I Base Integer.

4. Nios V/g Processor

The Nios V/g processor is a general-purpose CPU core developed by Altera based on the RISC-V RV32IMZicsr_Zicbom instruction set (optionally with "F" and "Smclic" extension) and supports the functional units described in this document.

Related Information

Overview on page 4

4.1. Processor Performance Benchmarks

Table 93. Nios V/g Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software

Quartus Prime Edition	FPGA Used	OPN	f_{MAX} (MHz)	Logic Size	Architecture Performance	
					DMIPS/MHz Ratio	CoreMark/MHz Ratio
Quartus Prime Pro Edition	Cyclone 10	10CX220YF780I5G	208	1999 ALM	1.317	2.323
	Arria 10	10AS066N3F40E2SG	220	1993 ALM		
	Stratix 10	1SX280LU2F50E2VG	214	2222 ALM		
	Agilex 7	AGFB014R24AR0	277	2207 ALM		
	Agilex 5	A5EC065BB32AE4S	239	2143 ALM		
Quartus Prime Standard Edition	Cyclone IV E	EP4CE115F29I8L	81	4316 LE	0.942	1.49
	Cyclone V	5CGTFD9E5F35C7	117	1886 ALM		
	Arria V	5AGXMB7G6F35C6	121	1917 ALM		
	Arria V GZ	5AGZME7K2F40C3	211	1859 ALM		
	Stratix V	5SGXEA7K2F40C2	231	1853 ALM		
	Cyclone 10 LP	10CL120YF780I7G	93	4174 LE		
	Arria 10	10AS066N3F40E2SG	239	1818 ALM		
	MAX 10	10M50DAF484C7G	91	4199 LE		

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Table 94. Benchmark Parameters for Quartus Prime Software

Parameter		Settings/Description	
		Quartus Prime Pro Edition	Quartus Prime Standard Edition
Quartus Prime seed		Maximum performance result are based on 10 seed sweep from Quartus Prime Pro Edition software version 25.1.	Maximum performance result are based on 10 seed sweep from Quartus Prime Standard Edition software version 23.1.
Device speed grade		Fastest speed grade from each Altera FPGA device family.	
Defined peripherals		• Nios V/g processor core • Without debug module, internal timer, and floating point unit • With 4 KB instruction cache, 4 KB data cache, and branch prediction. • 128 KB on-chip memory for the instruction and data bus. • JTAG UART Intel FPGA IP. • Interval Timer Core.	
Toolchain	Version	• riscv32-unknown-elf-gcc (GCC) version 13.2.0 • CMake Version: 3.31.3	• riscv32-unknown-elf-gcc (GCC) version 12.1.0 • CMake Version: 3.27.1
	Compiler configuration	• Compiler flags: `-O3` • Assembler options: `-Wa -gdwarf2` • Compile options: `-Wall -Wformat-security -march=rv32im_zicbom -mabi=ilp32`	

Altera uses the same Quartus Prime design example for maximum performance benchmark(fMAX) and logic size benchmarks. However, the compiler settings are different for each benchmarks:

- **Superior Performance with Maximum Placement Effort** in Quartus Prime Pro Edition software.
- **High Performance Effort** in Quartus Prime Standard Edition software.
- Logic size benchmark: `area_aggressive`

Note: Results may vary depending on the version of the Quartus Prime software, the version of the Nios V processor, compiler version, target device and the configuration of the processor. Additionally, any changes to the system logic design might change the performance and LE usage. All results are generated from design built with Platform Designer.

4.2. Processor Pipeline

The Nios V/g processor employs a five-stage pipeline.

Table 95. Processor Pipeline Stages

Stage	Denotation	Function
F	Instruction fetch	• PC+4 calculation • Next instruction fetch • Pre-decode for register file read
D	Instruction decode	• Decode the instruction • Register file read data available • Hazard resolution and data forwarding

continued...

Stage	Denotation	Function
E	Instruction execute	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
M	Memory	• Memory and multicycle operations • Register file write • Branch redirection
W	Write back	• Facilitates data dependency resolution by providing general-purpose register value.

The Nios V/g processor implements the general-purpose register file using the M20K memory blocks. The processor takes one processing cycle to read from an M20K location. Therefore, the F-stage initiates register file reads so general-purpose register values are available in D-stage.

Writing to the M20K location takes two processing cycles. Therefore, the M-stage initiates writes to a general-purpose register. If there is a dependency to resolve, the M-stage carries forward the value to the W-stage.

The core resolves data dependencies in the D-stage. Operands can move from register file read or E-stage, M-stage, or W-stage.

Reasons for the pipeline stalling:

- Data dependency—if the source operand is not available in D-stage, instruction in D-stage and F-stage stalls until the operand becomes available. The scenario can happen if destination general-purpose register of load or multicycle instruction in E-stage or M-stage is the source for instruction in D-stage.
- Resource stall—if a memory operation or multicycle is pending in M-stage, the instructions in preceding stages stalls until M-stage completes the instruction.

4.3. Processor Architecture

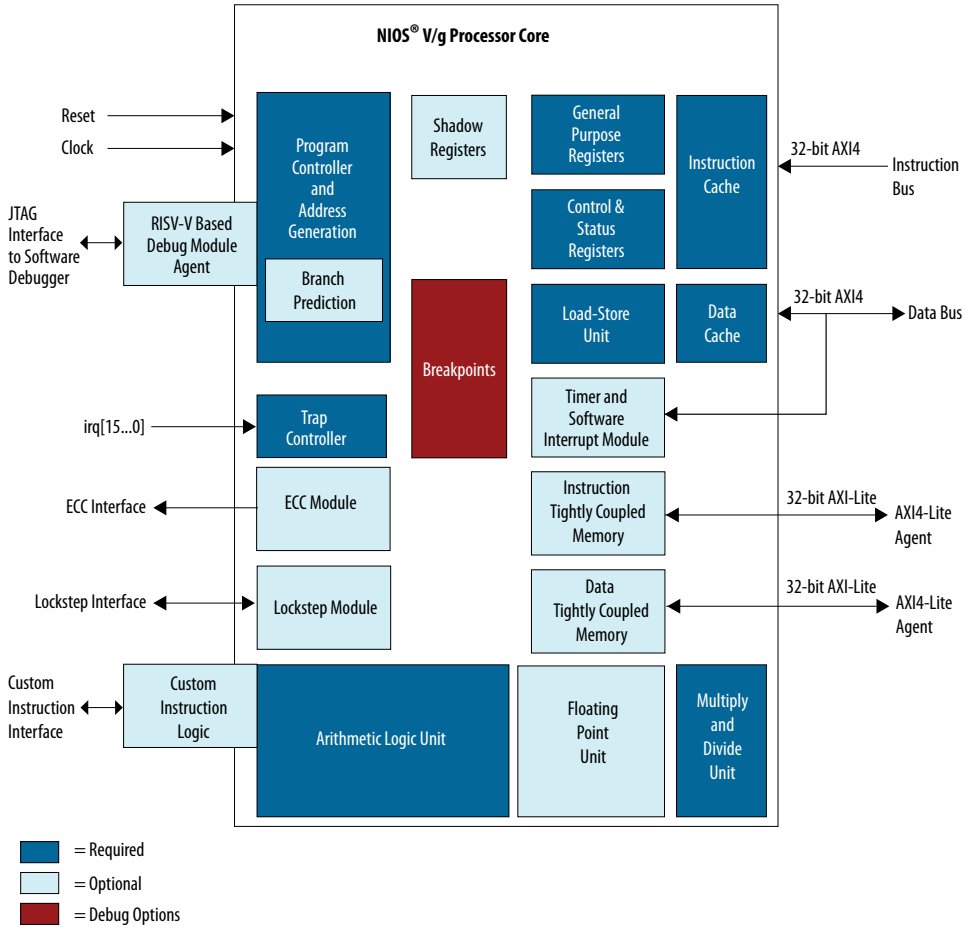
The Nios V/g processor architecture describes an instruction-set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions.

The Nios V/g processor architecture defines the following functional units:

- General-purpose register file
- Shadow register
- Arithmetic logic unit (ALU)
- Multiply and divide units
- Floating point unit
- Custom instruction logic
- Control and status registers (CSR)
- Trap controller
- Instruction bus
- Data bus
- Instruction cache

- Data cache
- Tightly coupled memories
- RISC-V based debug module
- ECC module
- Branch prediction
- Lockstep module

Figure 10. Nios V/g Processor Core Block Diagram



4.3.1. General-Purpose Register File

Nios V/g processor implementation supports a flat register file. The register file contains thirty-two 32-bit general-purpose integer registers. Nios V/g processor implements the general-purpose register using M20K memories, which do not support two read ports. Hence, Nios V/g processor duplicates the register files so that two different source registers for an instruction are available in a single cycle. After performing ALU operations, the processor core writes the same result to the destination register in both memories.

Table 96. Nios V Processor General-Purpose Register File

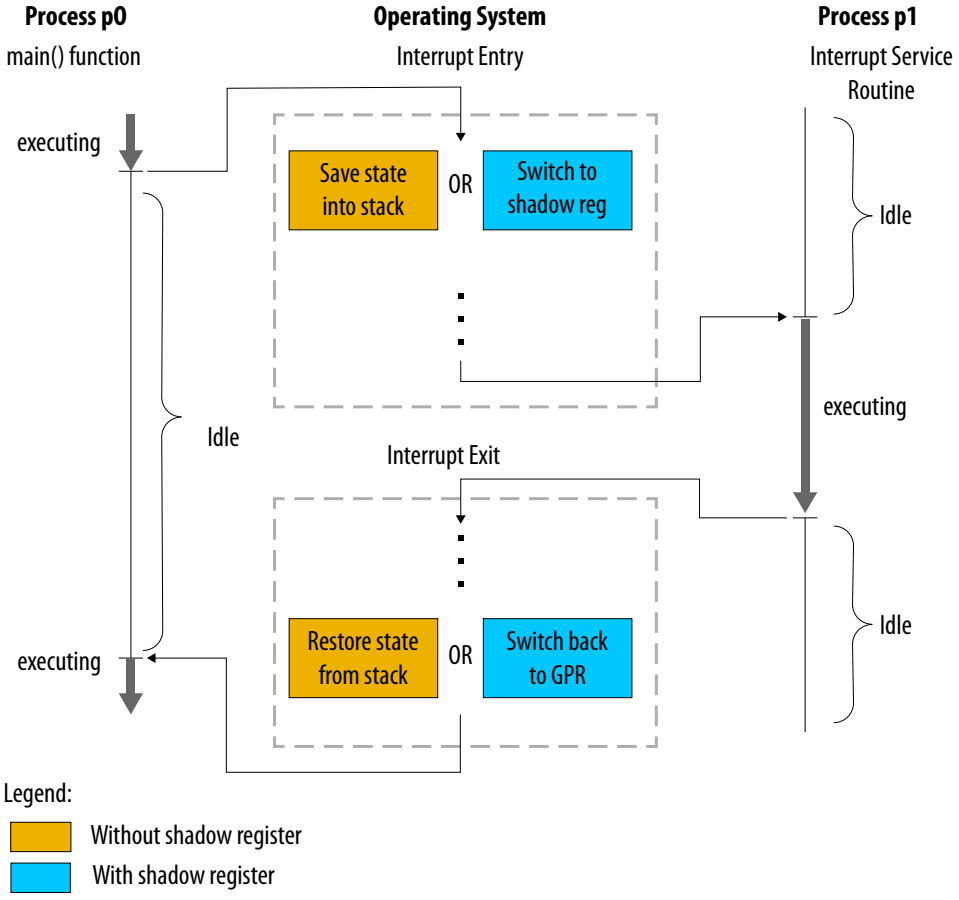
Register	ABI	Description
x0	zero	Hardwired to zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary 0
x6	t1	Temporary 1
x7	t2	Temporary 2
x8	s0/fp	• Saved register 0 • Frame pointer
x9	s1	Saved register 1
x10	a0	• Function argument 0 • Return value 0
x11	a1	• Function argument 1 • Return value 1
x12	a2	Function argument 2
x13	a3	Function argument 3
x14	a4	Function argument 4
x15	a5	Function argument 5
x16	a6	Function argument 6
x17	a7	Function argument 7
x18	s2	Saved register 2
x19	s3	Saved register 3
x20	s4	Saved register 4
x21	s5	Saved register 5
x22	s6	Saved register 6
x23	s7	Saved register 7
x24	s8	Saved register 8
x25	s9	Saved register 9
x26	s10	Saved register 10
x27	s11	Saved register 11
x28	t3	Temporary 3
x29	t4	Temporary 4
x30	t5	Temporary 5
x31	t6	Temporary 6

4.3.2. Shadow Register

The Nios V processor can have one or more optional shadow register sets. A shadow register set is an alternate set of Nios V general-purpose registers that works just like the general-purpose register set. This feature helps to accelerate context switching by swapping register file sets.

By maintaining a separate runtime context for an interrupt service routine (ISR), the processor software can remove the requirement to save and restore register state when entering or exiting an ISR respectively. This can greatly improve ISR performance and latency for timing critical ISRs.

Figure 11. Context Switching



Note that there is a total of 32 Nios V general-purpose registers, but the shadow register set comprised of the upper 16 registers only. Consequently, the shadow register set is not a complete alternate set of GPR. This implementation is intentionally for the following reasons:

- The RISC-V ABI assigns zero to x0 and pointer registers to x2 – x4. These registers must be in sync across all register sets.
- Reduce memory block utilization.

Besides GPRs, the Nios V/g processor support 32 floating-point registers, when floating-point unit is enabled. Unlike GPRs, the processor implements the shadow register as a complete alternate set of floating-point registers.

Enabling shadow register is backward compatible with ISRs that are built with shadow register disabled. To take advantage of the shadow registers, the ISRs can be enhanced by stop saving and restoring the upper 16 register state and floating-point registers.

Table 97. RV32I and "F" extension Register Set vs Shadow Register Set

RISC-V ISAa	Register Set	Saving and Restoring Registers during Context Switching		
		Shadow Register Set	Without Shadow Register	With Shadow Register
RV32I	x0 – x15	Not present	Required	Required
	x16 – x31	x16 – x31	Required	Not required (Replaced by register switching)
"F" Extension	f0 – f31	f0 – f31	Required	Not required (Replaced by register switching)

Note: The Altera HAL firmware is not optimized to use shadow register. All registers are saved and restored during context switching, despite that shadow register is implemented. Thus, interrupt response time remains the same, when running Altera HAL firmware with or without shadow register.

4.3.2.1. Shadow Register Implementation

Implementation of shadow register is affected by the presence of pre-emption. Without pre-emption in CLINT, the processor accepts a single interrupt only at any given time. On the other hand, pre-emption in CLIC enables processor to prioritize interrupts, thus requiring a new shadow register set to service the new pre-empting interrupts.

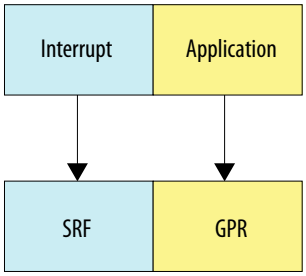
Depending on the trap controller, the Nios V processor core implements shadow register sets differently:

- CLINT-Direct or CLINT-Vectored – A single common shadow register set
- CLIC :
  - One shadow register set per interrupt level supported (**Number of CLIC interrupt levels**), or
  - One shadow register set per pre-empting interrupt level supported (**Number of CLIC interrupt levels- 1**)

4.3.2.1.1. CLINT

Without pre-emption (CLINT), the processor accepts a single interrupt only at any given time. Hence, a single common shadow register set (SRF) is sufficient.

Figure 12. Shadow Register Implementation in CLINT-Direct or CLINT-Vectored



4.3.2.1.2. CLIC with Number of CLIC interrupt levels option

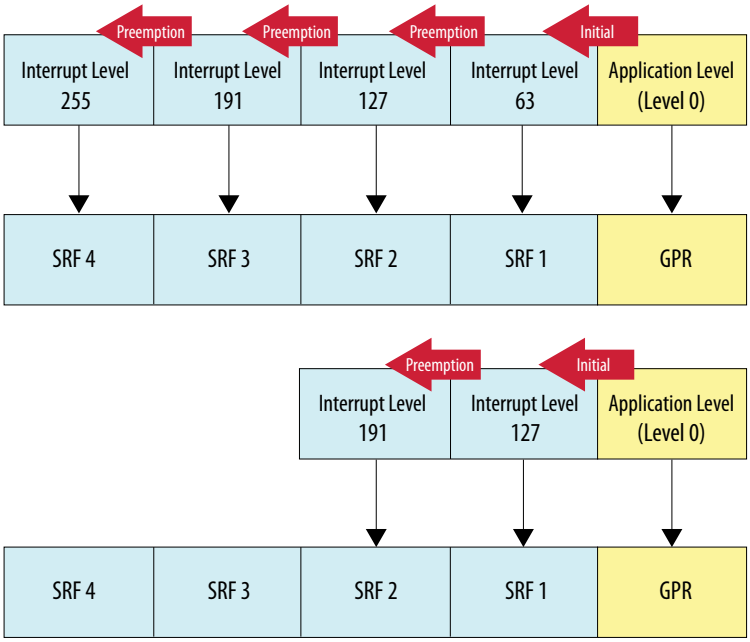
Pre-emption in CLIC enables processor to prioritize interrupts, requiring a new shadow register set to service the new interrupt. As such, each interrupt level has its own Shadow Register File (SRF) for handling interrupts, except for the Application Level, which uses the General-Purpose Register (GPR).

For example, if the processor supports 5 interrupt levels, you have:

- Application level 0
- Actual interrupt level 63, 127, 191 and 255

When you configure the Shadow Register Files to **Number of CLIC interrupt levels**, the system utilizes four SRFs and one GPR. There are five register sets, which translates to the "5" concept. The Nios V processor allocates the shadow register set to interrupt handlers using a "first come, first serve" basis starting from SRF1, while the Application Level uses the GPR.

Figure 13. Shadow Register in CLIC with Number of CLIC interrupt levels option



4.3.2.1.3. CLIC with Number of CLIC interrupt levels – 1 Option

While the **Number of CLIC interrupt levels** represents a conventional approach of dedicated register files for each level, **Number of CLIC interrupt levels – 1** offers an optimized approach that efficiently uses memory blocks. The **Number of CLIC interrupt levels** approach requires an odd number of register files. Implementing **Number of CLIC interrupt levels – 1** approach can reduce it by 1 to an even number, which fits perfectly into memory block as such:

- Eight register file sets into M20K
- Four register file sets into M9K

Similar as the example from before, if the processor supports five interrupt levels, you have:

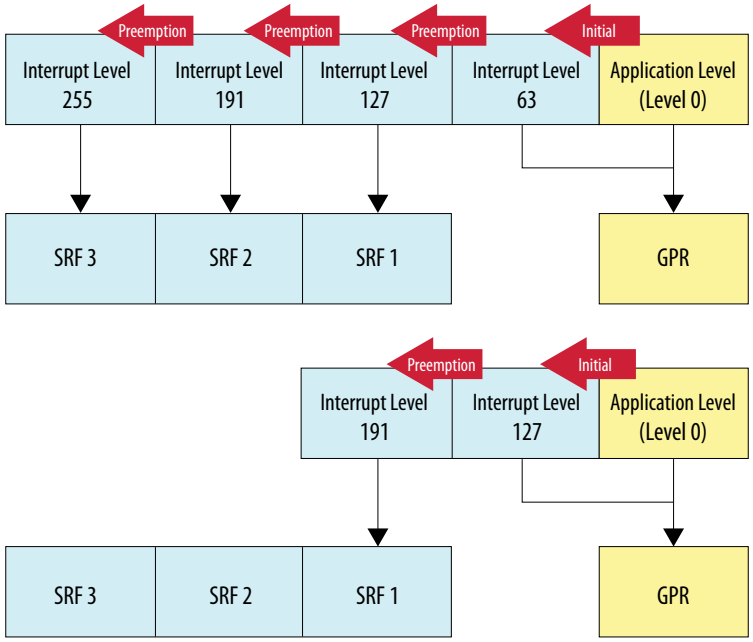
- Application level 0
- Interrupt level 63, 127, 191 and 255

When Shadow Register Files is configured to **Number of CLIC interrupt levels – 1**, three SRFs and one GPR are used. There are 4 register sets, which collates to "5 minus 1 = 4" concept. Nios V processor allocates shadow register set to pre-empting interrupt handlers only using "first come first serve" basis starting from SRF1, while initial interrupt and Application Level share a single GPR.

The table below displays an example of M9K utilization between the two approaches.

Shadow Register Implementation	Number of CLIC Interrupt Levels	Number of SRF	Number of GPR	Total Register Sets	Number of M9K Needed
Number of CLIC interrupt levels	5	4	1	5	2 M9K
Number of CLIC interrupt levels - 1	5	3	1	4	1 M9K

Figure 14. Shadow Register in CLIC with Number of CLIC interrupt levels – 1 option



4.3.2.2. Shadow Register Set Management

The Nios V processor uniquely supports the shadow register set, which is not found in any RISC-V specifications. Because of this, custom CSRs are needed to manage the shadow register set. You can access these custom CSR with the RISC-V `csrr*` instruction.

Enabling the shadow register set in the Nios V processor includes the addition of three new CSRs:

1. Shadow Register File Status CSR (`msrfstatus`)
2. Read Previous Shadow Register CSR (`mrdpsrf`)
3. Write Previous Shadow Register CSR (`mwrpsrf`)

4.3.2.2.1. Shadow Register File Status CSR (msrfstatus)

The `msrfstatus` is a 32-bits wide register containing shadow register status information. The `msrfstatus.ESI` bit determines if the hardware switches the shadow register files automatically upon taking an interrupt. This bit is cleared upon a processor reset.

Table 98. `msrfstatus` Register Layout

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

Table 99. `msrfstatus` Field Encoding

Bits	Field	Description
31	`ESI`	0: Disable register set switching upon interrupt is taken 1: Enable register set switching upon interrupt is taken
30:16	`Reserved`	-
15:8	`psrf`	Previous register file 0: General Purpose Register 1– 256: Shadow Register Sets
7:0	`asrf`	Active register file 0: General Purpose Register 1– 256: Shadow Register Sets

Table 100. `msrfstatus` Behaviour during Interrupt Handling (With `msrfstatus.ESI` bit = 1)

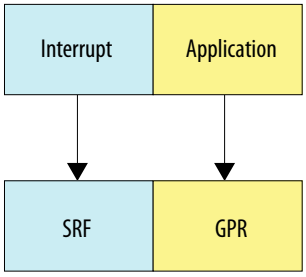
Interrupt Controller	Condition	msrfstatus.psrf	msrfstatus.asrf
CLINT	No interrupt	Don't Care	0 (General-purpose register)
	Typical interrupt entry	0 (General-purpose register)	1 (Shadow register)

continued...

Interrupt Controller	Condition	msrfstatus.psrf	msrfstatus.asrf
	Typical interrupt exit (back to No interrupt)	Don't Care	0 (General-purpose register)
CLIC + "Number of CLIC interrupt levels" option	No interrupt	Don't Care	0 (General-purpose register)
	Initial interrupt entry	0 (General-purpose register)	Any non-zero value (Shadow register) that equals to the initial interrupt level
	Pre-emption entry	Any non-zero value (Shadow register) that equals to the: • Initial interrupt level • Lower interrupt level	Any non-zero value (Shadow register) that equals to the higher interrupt level
	2nd or higher level pre-emption exit	Any non-zero value (Shadow register) that equals to a much lower interrupt level	Any non-zero value (Shadow register) that equals to the lower interrupt level
	1st pre-emption exit (back to Initial interrupt entry)	0 (General-purpose register)	Any non-zero value (Shadow register) that equals to the initial interrupt level
	Initial interrupt exit (back to No interrupt)	Don't Care	0 (General-purpose register)
CLIC + "Number of CLIC interrupt levels - 1" option	No interrupt	Don't Care	0 (General-purpose register)
	Initial interrupt entry	0 (General-purpose register)	0 (General-purpose register)
	1st pre-emption entry	0 (General-purpose register)	Any non-zero value (Shadow register) that equals to the 1st pre-empting interrupt level
	2nd or higher level pre-emption entry	Any non-zero value (Shadow register) that equals to the: • 1st pre-empting interrupt level • Lower interrupt level	Any non-zero value (Shadow register) that equals to the higher interrupt level
	3rd or higher level pre-emption exit	Any non-zero value (Shadow register) that equals to a much lower interrupt level	Any non-zero value (Shadow register) that equals to the lower interrupt level
	2nd pre-emption exit (back to 1st pre-emption entry)	0 (General-purpose register)	Any non-zero value (Shadow register) that equals to the 1st pre-empting interrupt level
	1st pre-emption exit (back to Initial interrupt entry)	0 (General-purpose register)	0 (General-purpose register)
	Initial interrupt exit (back to No interrupt)	Don't Care	0 (General-purpose register)

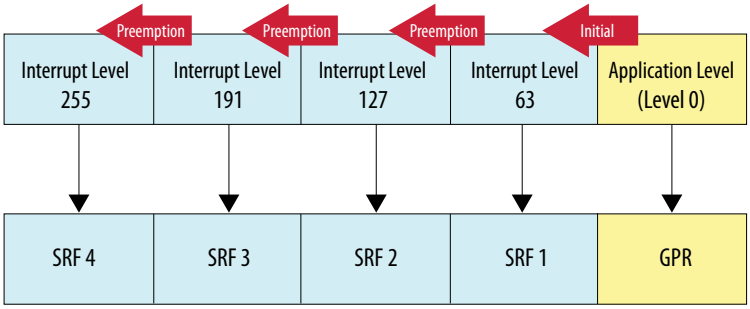
Note: When `msrfstatus .ESI` bit = 0, you need to update `msrfstatus.psrf` and `msrfstatus.asrf` manually using `csrr*` instruction to switch between register sets manually.

Figure 15. `msrfstatus` Behaviour in CLINT



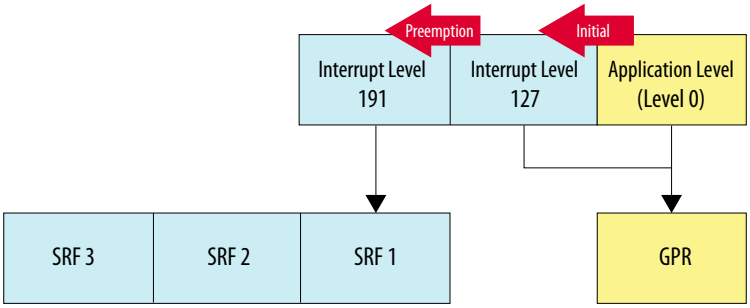
Events	msrfstatus CSR		Which processor state is held?	
	psrf	asrf	GPR	SRF
No interrupt	Don't Care	0	Application	-
Typical interrupt entry	0	1	Application	Interrupt
Typical interrupt exit	Don't Care	0	Application	-

Figure 16. `msrfstatus` Behaviour in CLIC with Number of CLIC interrupt levels Option



Events	msrfstatus CSR		Which processor state is held?				
	psrf	asrf	GPR	SRF 1	SRF 2	SRF 3	SRF 4
No interrupt	Don't Care	0	Level 0	-	-	-	-
Initial interrupt entry	0	63	Level 0	Level 63	-	-	-
1st Pre-emption entry	63	127	Level 0	Level 63	Level 127	-	-
2nd Pre-emption entry	127	191	Level 0	Level 63	Level 127	Level 191	-
3rd Pre-emption entry	191	255	Level 0	Level 63	Level 127	Level 191	Level 255
3rd Pre-emption exit	127	191	Level 0	Level 63	Level 127	Level 191	-
2nd Pre-emption exit	63	127	Level 0	Level 63	Level 127	-	-
1st Pre-emption exit	0	63	Level 0	Level 63	-	-	-
Initial interrupt exit	Don't Care	0	Level 0	-	-	-	-

Figure 17. `msrfstatus` Behaviour in CLIC with Number of CLIC interrupt levels – 1 Option



Events	msrfstatus CSR		Which processor state is held?			
	psrf	asrf	GPR	SRF 1	SRF 2	SRF 3
No interrupt	Don't Care	0	Level 0	-	-	-
Initial interrupt entry	0	0	Level 127	-	-	-
1st Pre-emption entry	0	191	Level 127	Level 191	-	-
1st Pre-emption exit	0	0	Level 127	-	-	-
Initial interrupt exit	Don't Care	0	Level 0	-	-	-

4.3.2.2.2. Read Previous Shadow Register CSR (`mrdpsrf`)

The `mrdpsrf` register facilitates copying a single register from previous shadow register sets to the active one. A csrrw instruction to `mrdpsrf` reads the rs1 register in previous register file and stores its value in the rd register of active register file.

For example: csrrw x10, mrdpsrf, x10

Read the value of x10 register in previous register file, and store it to x10 register of active register file.

4.3.2.2.3. Write Previous Shadow Register CSR (mwrpsrf)

The `mwrpsrf` register facilitates copying information from active shadow register sets to the previous one. A csrrw instruction to `mwrpsrf` reads the rs1 register in active register file and stores its value in the rd register of previous register file.

For example: csrrw x10, mwrpsrf, x11.

Read the value of x10 register in active register file, and store it to x11 register of previous register file.

4.3.3. Arithmetic Logic Unit

The arithmetic logic unit (ALU) operates on data stored in general-purpose registers. ALU operations take one or two inputs from registers and store the result back into the register.

Table 101. Fundamental Data Operations of the ALU

Category	Description
Arithmetic	Addition and subtraction on signed and unsigned operands.
Relational	Equal, not-equal, greater-than-or-equal, and less-than relational operations (==, != >=, <).
Logical	AND, OR, NOR, and XOR logical operations.
Shift	Logical and arithmetic shift operations.

For load and store instructions, the Nios V/g processor uses the ALU to calculate the memory address. For conditional control transfer instructions, Nios V/g processor uses the relational operations in the ALU to determine if the processor takes or leaves the branch.

4.3.4. Multipy and Divide Units

The multiply and divide units take two inputs from registers, compute, and store the result to the register.

4.3.5. Floating-Point Unit

The floating-point unit (FPU) implements the single precision floating point instructions. The FPU operates on data stored in thirty-two 32-bits floating-point registers, implemented using M20K memories.

Below are the characteristics of the FPU:

- Based on RISC-V "F" Standard Extension for Single-Precision Floating-Point
- Supports floating-point fused multiply-add instructions.
- IEEE 754-2008 compliant except for:
  - Simplified rounding
  - Subnormal supported on a subset of operations
- Consumes resource in a typical system as below(1):
  - 960 ALMs
  - Five M20Ks memories
  - Five DSP blocks

Note: The Nios V/g processor adopts the GNU floating point software emulation for double precision floating point operation.

Table 102. Nios V Processor Floating-Point Register File

Register	ABI	Description	Register	ABI	Description
f0	ft0	FP Temporary 0	f16	fa6	FP Argument 6
f1	ft1	FP Temporary 1	f17	fa7	FP Argument 7
f2	ft2	FP Temporary 2	f18	fs2	FP Saved register 2

continued...

(1) System using Arria 10 FPGA devices.

Register	ABI	Description	Register	ABI	Description
f3	ft3	FP Temporary 3	f19	fs3	FP Saved register 3
f4	ft4	FP Temporary 4	f20	fs4	FP Saved register 4
f5	ft5	FP Temporary 5	f21	fs5	FP Saved register 5
f6	ft6	FP Temporary 6	f22	fs6	FP Saved register 6
f7	ft7	FP Temporary 7	f23	fs7	FP Saved register 7
f8	fs0	FP Saved register 0	f24	fs8	FP Saved register 8
f9	fs1	FP Saved register 1	f25	fs9	FP Saved register 9
f10	fa0	• FP Argument 0 • FP Return value 0	f26	fs10	FP Saved register 10
f11	fa1	• FP Argument 1 • FP Return value 1	f27	fs11	FP Saved register 11
f12	fa2	FP Argument 2	f28	ft8	FP Temporary 8
f13	fa3	FP Argument 3	f29	ft9	FP Temporary 9
f14	fa4	FP Argument 4	f30	ft10	FP Temporary 10
f15	fa5	FP Argument 5	f31	ft11	FP Temporary 11

4.3.5.1. IEEE 754 Exception Conditions

The FPU offers different IEEE 754 exceptions support when targeting on different Altera FPGA devices. The `fcsr` register holds the accrued exception flags.

Table 103. IEEE 754 Exception Support

IEEE 754 Exception	Arria 10, Cyclone 10 GX	Other Altera FPGA devices
Invalid (NV)	√	√
Division by Zero (DZ)	√	√
Overflow (OF)	√	√
Underflow (UF)	Incomplete	√
Inexact (NX)	Incomplete	√

Related Information

Control and Status Register Field (CLINT) on page 131
Refer to the Floating-Point CSR Register Fields table for more information.

4.3.5.2. Floating Point Operations

The table below provides a detailed summary of the FPU operations.

Table 104. Floating Point Operation Summary

Category	Operation	Cycles[2]	Result	Subnormal	Rounding[3]	GCC Inference
Arithmetic	FDIV.S	14	a ÷ b	Flush-to-0	RNE	a / b
	FSUB.S	1	a - b	Flush-to-0	RNE	a - b
	FADD.S	1	a + b	Flush-to-0	RNE	a + b
	FMUL.S	2	a x b	Flush-to-0	RNE	a * b
	FSQRT.S	12	√a	Flush-to-0	Faithful [4]	sqrt(a)
	FMIN.S	2	(a < b) ? a : b	Supported	RNE	fminf()
	FMAX.S	2	(a < b) ? b : a	Supported	RNE	fmaxf()
Fused Arithmetic[5]	FMADD.S	3	(a x b) + c	Flush-to-0	RNE	(a * b) + c
	FMSUB.S	3	(a x b) – c	Flush-to-0	RNE	(a * b) - c
	FNMSUB.S	3	-(a x b) + c	Flush-to-0	RNE	-(a * b) + c
	FNMADD.S	3	-(a x b) - c	Flush-to-0	RNE	-(a * b) - c
Conversion	FCVT.S.W / FCVT.S.WU	3	int_to_float(a)	Supported	None	Casting
	FCVT.W.S / FCVT.WU.S	3	float_to_int(a)	Supported	Round towards Zero	Casting
					Round to Nearest, ties to Max Magnitude	roundf(a)
Compare	FLT.S	1	(a < b) ? 1 : 0	Flush-to-0	RNE	a < b
	FLE.S	1	(a ≤ b) ? 1 : 0	Flush-to-0	RNE	a <= b
	FEQ.S	1	(a = b) ? 1 : 0	Flush-to-0	RNE	a == b
Sign Injection	FSGNJN.S (FNEG.S)	1	-a	Supported	RNE	-a
	FSGNJX.S (FABS.S)	1	|a|	Supported	RNE	fabsf(a)
Classification	FCLASS.S	2	Refer to topic *Floating Point Classification*.	Supported	None	fpclassify(a)

Note: Assume a, b, and c as single-precision floating point values. Nios V Processor Fused Arithmetic has a rounding stage between the multiplier and addition.

The following list describes the header in the table above:

(2) Preliminary results.

(3) Round-to-Nearest, ties to Even (RNE).

(4) Faithful rounding has a maximum error of 1 Unit of Least Precision (ULP) as compared to the 0.5 ULP in RNE. Faithful rounding is employed to save area and reduce the latency of FSQRT.S.

(5) GCC toolchain infers Fused Arithmetic when the optimization level is -O3 or higher.

- **Operation** —Provides the name of the floating-point operation. The names match the names of the corresponding RISC-V floating-point instructions.
- **Cycle** —Specifies the number of cycles it takes to execute the instruction.
- **Result**—Describes the computation performed by the operation.
- **Subnormal**—Describes how the operation treats subnormal inputs and subnormal outputs. Subnormals are numbers with a magnitude less than approximately 1.17549435082e-38.
- **Rounding** —Describes how the FPU rounds the result.
- **GCC Inference**—Shows the C code from which GCC infers the instruction operation.

When optimizing a Floating Point Unit (FPU) within a processor, the FSQRT (floating-point square root) and FDIV (floating-point division) operations are critical components. Because of their complexity and longer execution time, they can influence the maximum frequency (Fmax) of the FPU. Refer to the following guidelines to help you decide whether to enable or disable the FSQRT and FDIV.

Table 105. Recommendations to Enable or Disable the FSQRT and FDIV

Action	Guidelines
Enable	Enable the FQSRT and FDIV: • If your application frequently performs floating-point square root and floating-point division calculations • To prioritize the number of instructions executed per cycle
Disable	Disable the FQSRT and FDIV: • If your application rarely or never uses floating-point square root and floating-point division calculations • To prioritize logic utilization and achieve a higher Fmax • If you're using the Nios V processor in a MAX 10 FPGA

In summary, deciding whether to enable or disable FSQRT and FDIV in a Floating Point Unit (FPU) is a choice between two main goals. You can aim for a higher maximum frequency (Fmax) and lower logic usage, or you can focus on improving performance for floating-point operations. This decision should depend on the specific needs and limits of your application.

Related Information

Floating Point Classification on page 76

4.3.5.2.1. Floating Point Classification

The FCLASS.S instruction classifies the floating-point value into ten possible classifications specified in the following table and returns an integer value.

Table 106. Floating Point Classification

Classification	Description	Single Precision FP Representation			Return Integer Value
		Sign (1 bit)	Exponent (8 bit)	Mantissa (23 bit)	
-infinity	Negative infinity	1	0xFF	0	0
-normal	Negative normalized non-zero value	1	Any value (except values representing other classes)		1

continued...

Classification	Description	Single Precision FP Representation			Return Integer Value
		Sign (1 bit)	Exponent (8 bit)	Mantissa (23 bit)	
-subnormal	Negative denormalized value	1	0	Non-zero	2
-0	Negative zero value	1	0	0	3
+0	Positive zero value	0	0	0	4
+subnormal	Positive denormalized value	0	0	Non-zero	5
+normal	Positive normalized non-zero value	0	Any value (except values representing other classes)		6
+infinity	Positive infinity	0	0xFF	0	7
Signaling NaN	Signaling NaN	Don't Care	0xFF	0x1 to 0x3FFFFF	8
Quiet NaN	Quiet NaN	Don't Care	0xFF	0x400000 to 0x7FFFFF	9

4.3.6. Custom Instruction

The Nios V/g processor architecture supports user-defined custom instructions. The Nios V/g ALU connects directly to custom instruction logic, enabling you to implement operations in hardware that are accessed and used in the same way as native instructions.

Related Information

AN 977: Nios V Processor Custom Instruction

4.3.7. Instruction Cycles

The table below provides instruction cycles for all types of instructions

Table 107. Instruction Cycles

Instructions	Category	Cycles	Pipeline Flush
BEQ, BNE, BLT, BGE, BLTU, BGEU	**With Branch Prediction**		
	Branch (Correctly predicted, taken)	2	-
	Branch (Correctly predicted, not taken)		
	Branch (Mis-predicted)	4	Yes
	Without Branch Prediction		
	Branch Taken	4	Yes
	Branch Not Taken	1	-
LB, LH, LW, LBU, LHU	Load (Without AXI-4/Avalon-MM transfer)	1	-
	Load (With AXI-4/Avalon-MM transfer)	More than 1	-
SB, SH, SW	Store (Without AXI-4/Avalon-MM transfer)	1	-

continued...

Instructions	Category	Cycles	Pipeline Flush
	Store (With AXI-4/Avalon-MM transfer)	More than 1	-
ADD, SUB, ADDI	Arithmetic	1	-
MUL, MULH, MULHSU, MULHU	Multiply	1	-
DIV, DIVU, REM, REMU	Divide	32	-
SLL, SLLI, SRA, SRAI, SRL, SRLI	Shift	1	-
SLT, SLTU, SLTI, SLTIU	Compare	1	-
AND, OR, XOR, ANDI, ORI, XORI	Logic Operation	1	-
JAR, JALR	Jump	4	Yes
ECALL, EBREAK	Environment Call and Breakpoint	5	Yes
CSRRW, CSRRS, CSRRC, CSRRWI, CSRSI, CSRRCI	Control and Status Register	1	-
LUI, AUIPC, FENCE, FENCE.TSO, PAUSE, CBO.FLUSH.IX, CBO.FLUSH, CBO.INVAL.IX, CBO.INVAL	Others	1	-

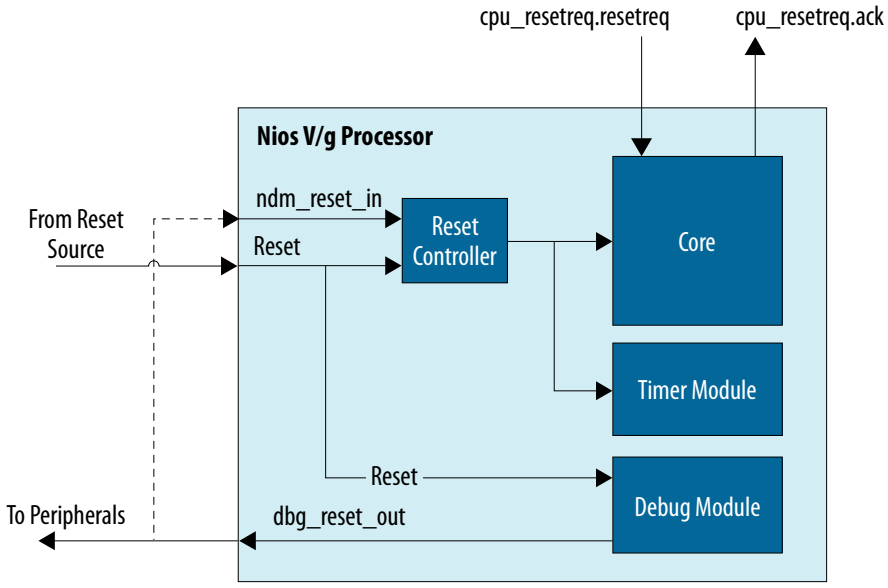
4.3.8. Reset and Debug Signals

Interface	Type	Description
`reset`	Reset	A global hardware reset input signal that forces the Nios V processor to reset immediately.
`dbg_reset_out`	Reset	An optional reset output signal which appear after you enable both **Enable Debug** and **Enable Reset from Debug Module** parameters. • This reset output signal is triggered by the JTAG debugger or `niosv-download -r` command. • You can connect this reset output signal to the following input signals: — To the `ndm_reset_in` input to reset the core and the timer module. — To the reset input signal of other components as needed.
`ndm_reset_in`	Reset	An optional reset input signal which appear after you enable both **Enable Debug** and **Enable Reset from Debug Module** parameters. • You can use this signal to trigger reset controller to reset the core and the timer module. • The reset controller synchronizes the reset (hard reset) and `ndm_reset_in` signals.
`cpu_resetreq`	Conduit	An optional local reset ports which appear after you enable **Add Reset Request Interface** parameter. The signal consists of an input `resetreq` signal and an output `ack` signal that trigger the Nios V processor to reset without affecting other components in a Nios V processor system.

continued...

Interface	Type	Description
		• You can request a reset to the Nios V processor core by asserting the `resetreq` signal. • The `resetreq` signal must remain asserted until the processor asserts `ack` signal. Failure for the signal to remain asserted can cause the processor to be in a non-deterministic state. • Assertion of the `resetreq` signal in debug mode has no effect on the processor's state. • The Nios V processor responds that the reset is successful by asserting the `ack` signal. • After the processor is successfully reset, the assertion of the ack signal can happen multiple times periodically until the de-assertion of the `resetreq` signal.

Figure 18. Nios V/g Processor Reset Network



4.3.9. Control and Status Registers

Nios V/g processor's Control and Status Registers (CSR) are both readable and writable. Nios V/g processor updates the CSR during the E-stage of the pipeline.

During the execution of a Nios V processor application, you may observe the following behaviors:

- CSR write instruction (in E-stage) is stalled due to the pending memory or multicycle instructions (in M-stage)
- CSR write instruction (in E-stage) continues after the pending instructions (in M-stage) are complete.
- If the processor generates an exception during the M-stage, the processor flushes the pending instructions in the pipeline (including the CSR write instruction in the E-stage) and initiates the trap handler to service the exception.

4.3.10. Trap Controller (CLINT)

In the Nios V processor, trap refers to the transfer of control to a trap handler caused by either an exception or an interrupt.

- Exceptions are synchronous events that originate inside the processor. They are commonly caused by an unusual condition occurring at run time associated with an instruction.
- Interrupts are asynchronous events that originate originated outside of the processor. They are commonly caused by service requests from system peripherals.

The RISC-V privileged ISA describes the supported trap modes and the control and status registers used and affected by trap handling. Therefore, this document contains information taken from the privileged ISA spec.

Related Information

RISC-V Specification
More information on RISC-V Trap Handling and related CSRs in The RISC-V Instruction Set Manual: Volume II - Privileged Architecture (Version 20240411).

4.3.10.1. Related Control and Status Register

Nios V processor supports only Machine-Level CSRs, which correlates with the fact that the Nios V processor supports Machine-mode (M-mode) only. Machine-Level CSRs serve as the primary privileged level and have low-level access to the machine implementation.

To simplify the following content on Nios V processor trap handling, CSRs or bit fields that require different privilege levels (e.g., User-mode and Supervisor-mode) are removed.

Since the Nios V processor supports M-mode only, the supported interrupts are Machine-level External (MEI), Machine-level Software (MSI), Machine Timer (MTI), and 16 Platform interrupts.

4.3.10.1.1. Machine Status Register (`mstatus`)

The fourth bit field of the status register (mstatus[3]) holds the global interrupt enable bit, MIE. This bit is commonly used to guarantee atomicity regarding interrupt handlers at the current privilege mode.

When a hart is executing, all interrupts are globally,

- Enabled when `MIE=1`, and
- Disabled when `MIE=0`.

An MRET instruction is used to return from a trap in M-mode.

4.3.10.1.2. Machine Trap-Vector Base-Address Register (mtvec)

Table 108. Machine Trap-Vector Base Address Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Base[31:2] (WARL)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base[31:2] (WARL)														Mode	

- The BASE field must always be 4-byte boundary aligned.
- The MODE field determines the processor's behaviour for asynchronous interrupts.

Table 109. Supported Vector Modes

mtvec	MODE	Description
00	Direct mode	All traps set pc to BASE << 2
01	Vectored mode	All exceptions set pc to BASE << 2 All interrupts set pc to BASE << 2+4*cause
10	Reserved	-
11	Reserved	-

When MODE=Direct ('00'), all traps into M-mode cause the program counter to be set to the address in the BASE field.

When MODE=Vectored ('01'), all synchronous exceptions cause the program counter to be set to the address in the BASE field, whilst asynchronous interrupts cause the program counter to be set to address in the BASE field plus four times the interrupt cause number.

4.3.10.1.3. Machine Interrupt Register (mip and mie)

The Machine Interrupt Pending register (mip) is a 32-bits wide register containing pending interrupts. The Machine Interrupt Enable register (mie) is a 32-bits wide register containing interrupt enable bits.

Each interrupt cause number corresponds to a bit in both mip and mie registers at bit location Exception Code in mcause register. Thus, interrupt n corresponds to bits mip[n] and mie[n].

Table 110. Machine Interrupt Pending register (mip) Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								MTIP	0			MSIP	0		

Table 111. Machine Interrupt Enable register (mie) Register Fields

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								MTIE	0			MSIE	0		

4.3.10.1.4. Machine Exception Program Counter Register (mepc)

The mepc register is a 32-bits wide read/write register. When a trap is taken into M-mode, mepc holds the virtual address of the instruction that was interrupted or encountered the exception. An MRET instruction jumps to the address stored in mepc.

4.3.10.1.5. Machine Cause Register (mcause)

Table 112. Machine Cause (mcause) Register Fields

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

The mcause register is a 32-bits wide register. When a trap is taken into M-mode, mcause holds a code identifying the event that caused the trap.

The Interrupt field is set when the trap was caused by an interrupt. The Cause field contains the code identifying the last trap event.

Table 113. mcause Value After Trap

Interrupt	Exception Code	Description
1	3	Machine software interrupt
1	7	Machine timer interrupt
1	16-31	16 Platform Interrupts, connected through irq[0:15] interface. For example: • Exception code 16 = irq[0] • Exception code 31 = irq[15]
1	Others	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal Instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault

continued...

Interrupt	Exception Code	Description
0	6	Store address misaligned
0	7	Store access fault
0	19	Hardware error due to uncorrectable ECC
0	Others	Reserved

If an instruction raises multiple exceptions, the decreasing priority order indicates which exception is taken and reported in `mcause`.

Table 114. Exception Priority in Decreasing Priority Order

Priority	Exception Code	Description
1 (Highest)	3	Instruction address breakpoint
2	1	With physical address for instruction: Instruction access fault
3	2 0 3 3	Illegal instruction Instruction address misaligned Environment break Load/Store address breakpoint
5	4,6	Optionally: Load/Store address misaligned
6	5,7	With physical address for an explicit memory access: Load/Store access fault
7 (Lowest)	4,6	Load/Store address misaligned

4.3.10.1.6. Machine Trap Value Register (`mtval`)

The `mtval` register is a 32-bits wide read/write register. When a trap is taken in M-mode, it is either set to zero or set to an exception specific value.

Table 115. `mtval` value After Exception

Exception Code	Exception Type	mtval value	mepc value
0	Instruction Address Misaligned	Faulting instruction physical address	Faulting instruction physical address
1	Instruction Access Fault		
4	Load Address Misaligned	Faulting load physical address	Interrupted instruction physical address
5	Load Access Fault		
6	Store Address Misaligned	Faulting store physical address	Interrupted instruction physical address
7	Store Access Fault		
3	Breakpoint	Interrupted instruction physical address	Interrupted instruction physical address

continued...

Exception Code	Exception Type	mtval value	mepc value
2	Illegal Instruction	Faulting instruction	Faulting instruction physical address
19	Hardware Error	• If ECC errors is outside of processor, faulting physical address • If ECC error is within processor, set to 0	Interrupted instruction physical address
-	Others	Set to 0	Interrupted instruction physical address

4.3.10.1.7. Machine Second Trap Value Register (`mtval2`)

The `mtval2` register is a 32-bits wide read/write register. When a trap is taken in M-mode, `mtval2` is written with additional exception specific information. The Nios V processor uses `mtval2` to provide additional information when an ECC induced Hardware Error exception occurs. Refer to the topic Error Correction Code) for more information.

4.3.10.2. Direct Mode

Direct Mode is a simple and efficient approach for trap controllers to manage interrupts and exceptions in embedded systems. In this method, each trap is assigned a fixed, specified memory address where the trap handling code is stored. When a trap occurs, the trap controller jumps to this address and executes the trap handling code.

The trap handling code:

- Switches to the separate exception stack (if enabled).
- Stores register values onto the stack.
- Determines the type of exception.
- Passes control to the exception dispatcher or the interrupt dispatcher, providing a quick and direct response to the event.

The Direct Mode supports interrupt pre-emption based on privilege mode, i.e., Machine, Supervisor, and User mode. Since the Nios V processor only supports Machine mode (M-mode), Direct Mode does not support interrupt pre-emption based on privilege mode.

4.3.10.2.1. Hardware Implementation

When a trap occurs, the trap controller recognizes the type of interrupt and directly switches the processor to the trap handling code by configuring the `pc` to the address specified in the BASE field of the `mtvec` register.

Table 116. Address Computation in Direct Nios V Processor Trap Controller

Assumed the BASE field in `mtvec` register is 0x20.

Trap	Address Computation	Description	Results
Exceptions	`pc <= BASE < < 2`	All exceptions and interrupts	`pc <= 0x80`
Interrupts			

4.3.10.2.2. Software Implementation

When a trap (either exception or interrupt) happens, the trap controller directs it through the trap handling code.

Related Information

Nios® V Processor Software Developer Handbook: Trap Handling

4.3.10.3. Vectored Mode

The Vectored Mode is designed to provide low-latency and vectored interrupts for RISC-V systems. Vectored Mode in RISC-V systems efficiently manages traps by directing the processor to the appropriate trap handler using a vector table. The trap controller uses the base address contained in the BASE field of `mtvec` register, to determine the address of the handler based on the `Interrupt` and `Exception Code` field from the mcause register. This method enables quick and precise redirection to the correct handler, improving flexibility and reaction time while managing various traps.

The Vectored Mode supports interrupt pre-emption based on privilege mode, i.e., Machine, Supervisor, and User mode. Since the Nios V processor only supports Machine mode (M-mode), Vectored Mode does not support interrupt pre-emption based on privilege mode.

Note: CLINT-Vectored interrupt mode is not supported in Nios V/m-Non-pipelined processor.

4.3.10.3.1. Hardware Implementation

In Vectored Mode, synchronous exceptions and asynchronous interrupts implement different address calculations. The Nios V processor determines whether the traps are exceptions or interrupts from the Interrupt field of `mcause` register.

For synchronous exceptions, such as illegal instructions, the hardware assigns the pc to the address provided in the BASE field of the `mtvec` register. This base address serves as a fixed entry point for the generic exception handler routine.

When an interrupt occurs, the system computes the next pc address to the base address from the BASE field plus four times the interrupt cause number, which is acquired from `Exception Code` in the `mcause` register. This calculation determines the address of the dedicated interrupt handler in the vector table.

Note: The effective range for jump instructions is limited to within 1 MiB from the current instruction's address.

Table 117. Address Computation in Vectored Nios V Trap Controller

Trap	Address Computation	Description	Exception Code	Results
Exceptions	`pc <= BASE << 2`	All exceptions	Starting from 0 (Instruction address misaligned), up to 19 (Hardware error)	`pc <= 0x80`
Interrupts	`pc <= BASE << 2 + 4*Exception Code`	Machine software interrupt	3	`pc <= 0x8C`

continued...

Trap	Address Computation	Description	Exception Code	Results
		Machine timer interrupt	7	`pc <= 0x9C`
		Platform interrupt 0	16	`pc <= 0xC0`
		…	…	…
		Platform interrupt 15	32	`pc <= 0x100`

Note: Assume the BASE field in mtvec register is 0x20.

4.3.10.3.2. Software Implementation

When a trap occurs, whether it is an exception or an interrupt, the trap controller directs it to either the exception dispatcher for exceptions or the interrupt service routine for interrupts.

Related Information

Nios® V Processor Software Developer Handbook: Trap Handling

4.3.11. Trap Controller (CLIC)

The Core-Local Interrupt Controller (CLIC) Privileged Architecture Extensions are designed to provide low-latency and pre-emptive interrupts for RISC-V systems.

The CLIC supports up to 2064 interrupt inputs per hart. The first 16 interrupt inputs are reserved for the original basic mode interrupts, so up to 2048 local platform interrupts can be added.

Note: Nios V processor CLIC does not support Selective Hardware Vectoring .

Note: RISC-V International has not yet ratified the CLIC specification.

4.3.11.1. Related Control and Status Register Fields

The Nios V processor only supports Machine-Level Control and Status Registers (CSRs) because it operates only in Machine Mode (M-mode). It is the primary privileged level and has low-level access to the machine implementation.

When you enable CLIC, it introduces new indirect access M-mode CSRs and new M-mode CSRs into the Nios V processor. Additionally, it changes some existing CSRs that were based on CLINT specifications to meet CLIC specifications.

Table 118. Machine Mode CSRs related to CLIC

Number	Name	From CLINT to CLIC	Description
`0x300`	`mstatus`	MODIFIED	Status register
`0x304`	`mie`	INACTIVE	Interrupt-enable register
`0x305`	`mtvec`	MODIFIED	Trap-handler base address / interrupt mode
`0x307`	`mtvt`	NEW	Trap-handler vector table base address
`0x340`	`mscratch`	NO CHANGE	Scratch register for trap handlers

continued...

Number	Name	From CLINT to CLIC	Description
0x341	mepc	NO CHANGE	Exception program counter
0x342	mcause	MODIFIED	Cause of trap
0x343	mtval	NO CHANGE	Bad address or instruction
0x344	mip	INACTIVE	Interrupt-pending register
0x345	mnxti	NEW	Interrupt handler address and enable modifier
0x347	mintthresh	NEW	Interrupt-level threshold
0x349	mscratchcswl	NEW	Conditional scratch swap on level change
0x350	miselect	NEW	Machine Indirect Select
0x351	mireg	NEW	Machine Indirect Alias
0x352	mireg2	NEW	Machine Indirect Alias 2
0xFB1	mintstatus	NEW	Current interrupt levels

In the Nios V processor, you can switch between CLINT and CLIC mode using the `mtvec` register. Thus, it is important to recognize which CSRs—CLINT CSRs or CLIC CSRs, are accessible after the switch is complete. Accessing CLIC CSRs in CLINT mode causes the processor to report illegal instruction exception, except for the `mintstatus` register. However, note that the `mintstatus` register is irrelevant in CLINT mode.

Related Information

Trap Controller (CLINT) on page 80
For more information about CLINT if the mtvec register is in CLINT mode.

4.3.11.1.1. CLIC Registers

The Nios V processor implements each interrupt input `i` with the following control registers:

- 8-bit slice of Interrupt Attribute (`clicintattr[i]`) that configures the trigger type,
- 8-bit slice of Interrupt Input Control (`clicintctl[i]`) that configures the interrupt's level and priority,
- 1-bit slice that holds interrupt-pending (`clicintip[i]`), and
- 1-bit slice that holds interrupt-enable bit (`clicintie[i]`).

Table 119. CLIC Registers

Name	Access	Description
clicintattr	Read/Write	CLIC Interrupt Attribute
clicintctl	Read/Write	CLIC Interrupt Input Control
clicintip	Read/Write	CLIC Interrupt Pending
clicintie	Read/Write	CLIC Interrupt Enable

If an interrupt `i` is not present in the hardware, the corresponding CLIC registers for interrupt `i` are hardwired to zero.

Access to CLIC registers – `clicintctl`, `clicintattr`, `clicintip`, and `clicintie` utilizes the 32-bit Indirect Access M-mode CSRs. For more information about the Indirect Access M-mode CSRs, please refer to the Indirect Access M-mode CSR chapter.

Note that the CLIC registers are not Nios V pocessor M-Mode CSRs.

- They are accessible using Indirect Access CSRs only.
- Direct access using CSR instruction is not possible.

CLIC Interrupt Attribute (`clicintattr`)

This is an 8-bit WARL read-write register to specify various attributes for each interrupt. You can configure the interrupt's trigger condition based on polarity and trigger.

There are four possible combinations of trigger conditions:

- Positive level-triggered
- Negative level-triggered
- Positive edge-triggered, or
- Negative edge-triggered

All interrupts in CLIC are software vectored because the Nios V processor does not support selective hardware vectoring.

Table 120. `clicintattr` Register Layout

7	6	5	3	2	1	0
mode = 11		Reserved		polarity	trigger	shv = 0

Table 121. Interrupt Attributes Field Encoding

Bits	`clicintattr` Field	Description
7:6	Mode	3: Machine mode
5:3	Reserved	-
2	Polarity	0: Positive Edge/High Level 1: Negative Edge/Low Level
1	Trigger	0: Level Triggered 1: Edge Triggered
0	Selective Hardware Vectoring	0: Software Vectored Mode

CLIC Interrupt Input Control (`clicintctl`)

`clicintctl[i]` is an 8-bit control register to specify interrupt level and interrupt priority. The most significant bit specifies the interrupt level while the least significant bit specifies the interrupt priority. If fewer than 8 bits are needed to specify interrupt and priority, the least significant bits of `clicintctl[i]` are WARL 1.

The processor selects the interrupt with the highest level by adhering to this procedure:

1. The CLIC hardware picks the global maximum across all pending-and-enabled interrupts based on `clicintctl`.
2. The interrupt level of this selected interrupt is compared with the interrupt-level threshold (`mintthresh.threshold`) to determine whether:
   - The level is qualified (Accepts interrupt), or
   - The level is masked (Ignores interrupt).

Phase 1: Assume the following settings is made in the Nios V processor IP Parameter Editor.

Table 122. Possible Interrupt Levels and Priorities based on Cases

Case	# of Levels Supported	# of Priorities Supported	Possible Interrupt Levels	Possible Interrupt Priorities
A	256 levels	1 priority	0, 1, 2, …, 253 , 254, 255	255
B	17 levels (16 levels + Level 0)	16 priorities	0, 15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255	15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255
C	3 levels (2 levels + Level 0)	128 priorities	0, 127, 255	1, 3, 5, …, 251, 253, 255
D	9 levels (8 levels + Level 0)	16 priorities	0, 31, 63, 95, 127, 159, 191, 223, 255	15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255
E	9 levels (8 levels + Level 0)	8 priorities	0, 31, 63, 95, 127, 159, 191, 223, 255	31, 63, 95, 127, 159, 191, 223, 255
F	2 levels (1 level + Level 0)	1 priority	0, 255	255

At any given time, the Nios V processor CLIC implements the specified number of interrupt levels plus an additional level 0 for application code. With non-zero interrupt levels as the only options for interrupts, the application code is always at a lower level than any interrupts.

Note: In Case A, the interrupt level 0 corresponds to the execution of application code, and must not be assigned to an interrupt. Assigning the interrupt level 0 disables the interrupt.

Phase 2: Writing `clicintctl` register with 0x85 or binary value of `10000101`,

Table 123. Configuration Made from Same Binary Values per Cases

Case	Valid bits in `clicintctl`		WARL bits	Configuration made for this interrupt	
	Level	Priority		Level	Priority
A	8 bits	0 bit	0	133 (binary = **10000101**)	255 (binary = 11111111)
B	4 bits	4 bits	0	143 (binary = **1000**1111)	95 (binary = **0101**1111)
C	1 bit	7 bits	0	255 (binary = **1**1111111)	11 (binary = **0000101**1)

continued...

Case	Valid bits in clicintctl		WARL bits	Configuration made for this interrupt	
	Level	Priority		Level	Priority
D	3 bits	4 bits	1 bit	159 (binary = **100**11111)	47 (binary = **0010**1111)
E	3 bits	3 bits	2 bits	159 (binary = **100**11111)	63 (binary = **001**11111)
F	0 bit	0 bit	8 bits	255 (binary = 11111111)	255 (binary = 11111111)

CLIC Interrupt Pending (clicintip)

Each interrupt input has a dedicated interrupt pending bit `clicintip[i]`, and `clicintip[i]` is unaffected by `clicintie[i]` setting. In CLIC, each interrupt is configurable as a level-sensitive interrupt or an edge-sensitive interrupt. Note that, the value in the `clicintip[i]` is undefined when switching from level-sensitive mode to edge-triggered mode in `clicintattr[i]`.

- When `clicintip[i]= 0`, there is no pending interrupt with respect to interrupt `i`.
- When `clicintip[i]= 1`, there is a pending interrupt with respect to interrupt `i`, and the processor is expected to service the interrupt request.

Level-Sensitive Inputs

When the input is configured for level-sensitive input, the `clicintip[i]` bit reflects the value of an input signal to the interrupt controller after any conditional inversion (specified by the `clicintattr[i]` field), and software writes to the bit are ignored. Software clears the interrupt at the source device.

Edge-Sensitive Inputs

When the input is configured for edge-sensitive input, `clicintip[i]` is set by CLIC after an edge of the appropriate polarity is observed on the interrupt input. After the edge-sensitive interrupt is serviced, clearance of the `clicintip[i]` depends on the type of interrupt:

- When a software vectored interrupt is serviced, the CLIC does not automatically clear the associated interrupt pending bit. It is cleared only after the execution of the appropriate access to the `mnxti` CSR (a CSR instruction that includes a write).
- When a hardware vectored interrupt is serviced, the CLIC automatically clears the associated interrupt pending bit.
- Alternatively, software writes to `clicintip[i]` can set or clear edge-triggered pending bits directly.

Note: All interrupts in CLIC are software vectored because Nios V processor does not support selective hardware vectoring.

CLIC Interrupt Enable clicintie

Each interrupt input has a dedicated interrupt-enable bit (`clicintie[i]`) and occupies 1-bit in the memory map for ease of access. This control bit is read-write to enable/disable the corresponding interrupt.

clicintie[i] is the individual enable bit while mstatus.mie is the global enable bit for the current privilege mode. Therefore, for an interrupt i to be enabled in the current privilege mode, you must set both clicintie[i] and mstatus.mie.

Note: Nios V processor hardwires clicintie[i] of unimplemented interrupts to zero.

4.3.11.1.2. Indirect Access M-mode CSRs

Access to CLIC registers—clicintctl, clicintattr, clicintip, and clicintie utilizes the 32-bit Indirect Access M-mode CSRs.

Table 124. Indirect Access M-mode CSRs

CSR Address	Name	Access	Description
0x350	miselect	WARL	Machine Indirect Select
0x351	mireg	Read/Write	Machine Indirect Alias
0x352	mireg2	Read/Write	Machine Indirect Alias 2

The CSRs listed in the table above provide a window for accessing register state indirectly. The value of miselect determines which register is accessed upon read or write of each of the machine indirect alias CSRs—mireg*.

Packing of Indirectly Accessed CLIC Registers

When the processor is accessing CLIC registers through Indirect Access CSR, the CLIC registers are packed together into 32-bit, as follows:

- Four 8-bits clicintctl slices
- Four 8-bits clicintattr slices
- Thirty-two 1-bit clicintip slices
- Thirty-two 1-bit clicintie slices

Thus, a single CSR Read/Write operation to Indirect Access CSR allows you to access the configuration and status of multiple interrupts, that are connected to CLIC.

Read/Write clicintctl[i] and clicintattr[i]

Based on the value of miselect register (Base value of 0x1000):

- The mireg register represents the clicintctl of four interrupts, which controls their respective CLIC level and priority.
- The mireg2 register represents the clicintattr of four interrupts, which controls their respective CLIC polarity and trigger.

Table 125. clicintctl and clicintattr Access using Indirect CSRs

miselect	Register	Bit Fields	Description
0x1000 + index	mireg	7:0	clicintctl of interrupt { index*4 + 0 }
		15:8	clicintctl of interrupt { index*4 + 1 }
		23:16	clicintctl of interrupt { index*4 + 2 }

continued...

miselect	Register	Bit Fields	Description
		31:24	clicintctl of interrupt { index*4 + 3 }
	mireg2	7:0	clicintattr of interrupt { index*4 + 0 }
		15:8	clicintattr of interrupt { index*4 + 1 }
		23:16	clicintattr of interrupt { index*4 + 2 }
		31:24	clicintattr of interrupt { index*4 + 3 }

Example 1. To configure Interrupt 7 (Machine Timer Interrupt), index equals to 1

- miselect equals to 0x1001.
- mireg[31:24] holds the clicintctl of Interrupt 7.
- mireg2[31:24] holds the clicintattr of Interrupt 7.

Example 2. To configure Interrupt 40 (Platform Interrupt 25), index equals to 10

- miselect equals to 0x100A.
- mireg[7:0] holds the clicintctl of Interrupt 40.
- mireg2[7:0] holds the clicintattr of Interrupt 40.

Read/Write clicintip[i] and clicintie[i]

Based on the value of miselect register (Base value of 0x1400):

- The mireg register represents the clicintip of 32 interrupts, which displays their interrupt pending state.
- The mireg2 register represents the clicintie of 32 interrupts, which displays their interrupt enable state.

Table 126. clicintip and clicintie Indirect CSRs

miselect	Register	Bit Field	Description
0x1400 + index	mireg	31:0	clicintip of interrupts from { index*32 } to { (index+1)*32 – 1 }
	mireg2	31:0	clicintie of interrupts from { index*32 } to { (index+1)*32 – 1 }

To view pending and enable status of Interrupt 7 (Machine Timer Interrupt), index equals to 0.

- miselect equals to 0x1400.
- mireg[7] holds the clicintip of Interrupt 7.
- mireg2[7] holds the clicintie of Interrupt 7.

To view pending and enable status of Interrupt 40 (Platform Interrupt 25), index equals to 1.

- miselect equals to 0x1401.
- mireg[8] holds the clicintip of Interrupt 40.
- mireg2[8] holds the clicintie of Interrupt 40.

4.3.11.1.3. New M-mode CSRs

Machine Trap-handler Vector Table base address Register (mtvt)

The mtvt 32-bit CSR holds the base address of the trap vector table, aligned on a 64-byte or greater power-of-two boundary. The actual alignment can be determined by writing ones to the low-order bits then reading them back. Values other than 0 in the low 6 bits of mtvt are reserved.

The value of the mtvt CSR is provided when the mnxti CSR is read.

Table 127. mtvt register layout

31	6	5	0
base[31:6]		Reserved	

Machine Next Interrupt Handler Address and Interrupt Enable Register (mnxti)

The mnxti CSR improves the performance of handling back-to-back software vectored interrupts. It does this by avoiding the overhead of additional interrupt pipeline flushes and redundant context save/restores for these back-to-back software vectored interrupts. The mnxti CSR is intended to be used inside an interrupt handler after an initial interrupt has been taken and mcause and mepc registers have been updated with the interrupted context and the ID of the initial interrupt.

The value returned by a CSR read of mnxti is the non-zero address of the CLIC vector table entry when there is a suitable pending interrupt. Otherwise, zero is returned. For a pending back-to-back interrupt to be considered, it must fulfil the following conditions:

- It is a software vectored interrupt
- It has a level greater than the saved interrupt level (held in mcause.mpil)
- It has a level greater than the interrupt threshold (held in mintthresh)

Based on the CSR instruction, the processor behaves differently when assessing mnxti CSR.

Table 128. Behaviours when assessing mnxti CSR

Processor Behaviours	Condition	List of CSR Instructions
1. Updates minstatus.mil to the level of new interrupt 2. Updates mcause.excode to ID of new interrupt 3. Clear the new interrupt clicintip bit (Only when edge-triggered) 4. Returns a pointer to the trap handler entry	CSR instruction includes write	• CSRRCI • CSRRS with zero immediate values for all bits, except bits 1 and 3 • CSRRSI with zero immediate values for all bits, except bits 1 and 3
Returns a pointer to the trap handler entry	CSR instruction does not include write	CSRR
Reserved	N/A	• CSRRW • CSRRWI • CSRRC • CSRRSI with non-zero immediate values for bits 0, 2 and 4

Machine Interrupt Status Register (mintstatus)

A new M-mode CSR, mintstatus, holds the active interrupt level for supported privilege mode. These fields are read-only. The primary reason to expose these fields is to support debugging.

Table 129. mintstatus Register Layout

31	24	23	16	15	8	7	0
mil		Reserved		0		Reserved	

The mintstatus registers are accessible in CLINT mode for processor systems that support both CLINT and CLIC modes.

Machine Interrupt-Level Threshold Register (mintthresh)

The interrupt-level threshold (mintthresh) is a new read-write CSR, which holds an 8-bit field (threshold) for the threshold level of the associated privilege mode. The threshold field is held in the least-significant 8 bits of the CSR, and zero should be written to the upper bits.

A typical usage of the interrupt-level threshold is for implementing critical sections. The current handler can temporarily raise its effective interrupt level to implement a critical section among a subset of levels, while still allowing higher interrupt levels to pre-empt.

Table 130. mintthresh Register Layout

31	8	7	0
Reserved (Hardwired as 0)		threshold (WARL)	

The current hart's effective interrupt level is effective_level = max(mintstatus.mil, mintthresh.th)

Machine Scratch Swap for Interrupt-Level Register (mscratchcswl)

A new mscratchcswl CSR is added to support conditional swapping of the stack pointer between interrupt and non-interrupt code running in the same privilege mode.

Common use case – csrrw sp, mscratchswl, sp

- Execute swap when accepting an interrupt from non-interrupt code, or returning to non-interrupt code from an interrupt.
- Only one of both mcause.mpil and mintstatus.mil is the application level 0.
- Otherwise, no action is taken.
- The CSR instruction is used once at the entry to a handler routine and once at handler exit, so only adds two instructions to the interrupt code path.

The mscratchcswl CSR is designed to be used with the csrrw instruction with neither rd nor rs1 set to x0. Refrain from accessing the mscratchcswl CSR with:

- The csrrw instruction with either rd or rs1 set to x0, or
- Using any other CSR instruction (CSRRWI/CSRRS/CSRRC/CSRRSI/CSRRCI).

4.3.11.1.4. Existing CSRs with Changes

Machine Status Register (mstatus)

When in CLIC mode, the mpp and mpie in mstatus are also accessible via fields in the mcause register.

Machine Interrupt Register (mie & mip)

The mie CSR appears hardwired to zero in CLIC mode, replaced by the CLIC interrupt enable register – clicintie[i]. Similar implementation on the mip CSR with the CLIC interrupt pending register – clicintip[i].

- Writes to mie/mip is ignored and does not trigger an illegal instruction exception.
- Reads to mie/mip always return zero in CLIC mode.

Machine Trap-Vector Base-Address Register (mtvec)

The CLIC interrupt-handling mode is encoded as a new state in the existing mtvec WARL register, where mtvec.mode is 11, and mtvec.submode are zero. The trap vector base address is specified as the upper 26 bits of mtvec (base) with six lower zero bits appended, which constrains alignment on a 64-byte or larger power-of-two boundary.

Table 131. CLIC mode mtvec register layout

31	6	5	2	1	0
Base (WARL)		submode (Hardwired to 0)		mode	

Note: Altera recommends limiting the alignment of mtvec to 64-byte boundaries for processor system that implements both CLIC and CLINT mode.

Table 132. Supported Vector Modes

mtvec.mode	Modes	Description
00	CLINT Direct mode	All traps set pc to mtvec[31:2]. Refer to Trap Controller (CLINT) chapter for more information.
01	CLINT Vectored mode	All exceptions set pc to mtvec[31:2]. All interrupts set pc to mtvec[31:2]+4*cause. Refer to Trap Controller (CLINT) chapter for more information.
10	Reserved	-
11	CLIC mode	All exceptions and software-vectored interrupts set pc to mtvec[31:6]. All hardware-vectored interrupts set pc to mtvec[31:6]+4*cause.

Machine Cause Register (mcause)

The mcause register is a 32-bits wide register. When a trap is taken, mcause holds a code identifying the event that caused the trap.

Table 133. Machine Trap Cause Register Fields (For CLIC mode)

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
interrupt	minhv	mpp[1:0] = 00		mpie	Reserved			mpil[7:0]							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				exccode[11:0]											

Table 134. Mcause Field Encoding

Bits	Field	Description
31	`interrupt`	0: Exception 1: Interrupt
30	`minvh`	0: mepc contains (virtual) address of instruction
29:28	`mpp[1:0]`	Previous privilege mode. Same as `mstatus.mpp`
27	`mpie`	Previous interrupt enable. Same as `mstatus.mpie`
26:24	`Reserved`	-
23:16	`mpil[7:0]`	Previous interrupt level
15:12	`Reserved`	-
11:0	`exccode[11:0]`	Exception/interrupt code

The `mcause.mpp` and `mcause.mpie` fields mirror the `mstatus.mpp` and `mstatus.mpie` fields to reduce context save/restore code. For backwards compatibility in implementations supporting both CLINT and CLIC modes, when switching to CLINT mode the new CLIC `mcause.mpil` state field is zeroed.

Table 135. Mcause.interupt and mcause.exccode Value After Trap

Interrupt	Exception Code	Description
1	3	Machine software interrupt
1	7	Machine timer interrupt
1	16-2064	2048 Platform interrupts, connected through irq[0:2047] interface. For example: • Exception code 16 = irq[0] • Exception code 2064 = irq[2047]
1	Others	Reserved
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal Instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault

continued...

Interrupt	Exception Code	Description
0	6	Store address misaligned
0	7	Store access fault
0	Others	Reserved

If an instruction may raise multiple synchronous exceptions, the decreasing priority order indicates which exception is taken and reported in `mcause`.

Table 136. Exception Priority in Decreasing Priority Order

Priority	Exception Code	Description
Highest	3	Instruction address breakpoint
	1	With physical address for instruction: Instruction access fault
	2 0 3 3	Illegal instruction Instruction address misaligned Environment break Load/Store address breakpoint
	4,6	Optionally: Load/Store address misaligned
	5,7	With physical address for an explicit memory access: Load/Store access fault
Lowest	4,6	Load/Store address misaligned

4.3.11.1.5. Existing CSRs without Changes

Machine Scratch Register (`mscratch`)

The `mscratch` register is 32-bits wide read/write general purpose register. You can use it as a temporary register in trap handlers, and it typically points to a machine mode hart-local context space

Machine Exception Program Counter Register (`mepc`)

The `mepc` register is a 32-bits wide read/write register. When a trap is taken into M-mode, `mepc` holds the virtual address of the instruction that was interrupted or encountered the exception. An MRET instruction jumps to the address stored in `mepc`.

Machine Trap Value Register (mtval)

The `mtval` register is a 32-bits wide read/write register. When a trap is taken in M-mode, it is either set to zero or set to an exception specific value.

Table 137. `Mtval` Value After Exception

Exception Code	Exception Type	`mtval` value	`mepc` value
0	Instruction Address Misaligned	Faulting instruction physical address	Faulting instruction physical address
1	Instruction Access Fault		

continued...

Exception Code	Exception Type	mtval value	mepc value
4	Load Address Misaligned	Faulting load physical address	Interrupted instruction physical l address
5	Load Access Fault		
6	Store Address Misaligned	Faulting store physical address	Interrupted instruction physical address
7	Store Access Fault		
3	Breakpoint	Interrupted instruction physical address	Interrupted instruction physical address
2	Illegal Instruction	Faulting instruction	Faulting instruction physical address
-	Others	Set to 0	Interrupted instruction physical address

4.3.11.2. Hardware Implementation

4.3.11.2.1. Asynchronous Interrupt Handling

Software-Vectored Interrupt Handling

When `clicintattr[i].shv` is set to 0, the associated interrupt is handled as software-vectored interrupts. When servicing a software-vectored interrupt, the CLIC switches the processor execution to the common trap handling code by configuring the pc to the address specified in the BASE field of the `mtvec` register.

This is a simple and efficient approach for trap controllers to manage software-vectored interrupts and exceptions in embedded systems. In this mode, exceptions and software-vectored interrupts are assigned to a fixed memory address where the common trap handling code resides.

The common trap handling code is expected to provide a quick and direct response to the event, by executing the following operations:

- Switches to the separate exception stack (if enabled)
- Stores register values onto the stack
- Determines the type of traps (either exceptions or interrupts)
- Passes control to the exception dispatcher, or the interrupt dispatcher
- Optionally, support back-to-back software-vectored interrupts (with mxnti register).

Table 138. Address Computation in CLIC Direct Handling

Assume `mtvec` register holds the value of 0x1003, thus the BASE field in mtvec register is 0x40.

Trap	Address Computation	Description	Results
Exceptions	`pc <= BASE << 6`	All exceptions and software-vectored interrupts	`pc <= 0x1000`
Software-vectored Interrupts			

CLIC extends interrupt pre-emption to support up to 256 interrupt levels for machine mode, where higher-numbered interrupt levels can pre-empt lower-numbered interrupt levels. Interrupt level 0 corresponds to regular execution outside of an interrupt handler. Refer to Trap Handling with Pre-emption for more information.

Hardware-Vectored Interrupt Handling

All interrupts in CLIC are software vectored because Nios V processor does not support selective hardware vectoring.

4.3.11.2.2. Synchronous Exception Handling

Unlike interrupts, exceptions are not supported with configurable exception level, priority and hardware vectoring. Here are related descriptions:

- Exception Level—They are serviced with the same interrupt level as the instruction that raised the exception.
- Exception Priority—Refer to Table: Exception Priority in Decreasing Priority Order. Note that exception priority is unrelated to the interrupt priorities, and it is enforced upon multiple exceptions only.
- Hardware Vectoring—Not supported. The Nios V processor handles exception similar to Software-Vectored Interrupt Handling. Refer to **Software-Vectored Interrupt Handling** for more information.

4.3.11.2.3. Trap Handling with Pre-emption

In comparison with CLINT, CLIC offers more capabilities in interrupt management. The CLIC extends interrupt pre-emption to support up to 256 interrupt levels, where higher-numbered interrupt levels can pre-empt lower-numbered interrupt levels. Interrupt level 0 corresponds to regular execution outside of an interrupt handler.

Under CLIC mode, the processor is running under a specific interrupt level at any time. The current interrupt level is made visible in the `mintstatus` register. Incoming interrupts with a higher interrupt level can pre-empt an active interrupt handler running at a lower interrupt level, when interrupts are globally enabled in this privilege mode (`mstatus.mie = 1`).

The global interrupt-enable (`mstatus.mie`) guarantee atomicity with respect to interrupt handlers. It functions as such:

- If `mstatus.mie` is clear, no interrupts will be taken.
- If `mie` is set, any pending-enabled interrupts at a higher interrupt level will pre-empt current execution and run the interrupt handler for the higher interrupt level. For software vectored interrupts this takes the form of a re-entrant call to the common trap handler.

The overall behaviour is summarized in the following table.

Table 139. Behaviour Between Normal Execution and Interrupt

Before		Interrupt from CLIC	After			Description
mstatus.mie	pc	Interrupt Level	Interrupt Level Threshold	pc	epc	
0	main	Don't Care	Don't Care	main	-	Interrupt is disabled. Normal execution cannot be interrupted.
1	main	0	Don't Care	main	-	No interrupt is present. Normal execution continues.
1	main	Lower than threshold (mintthresh.th)	mintthresh.th	main	-	Interrupt level is lower than threshold. Interrupt is ignored, and normal execution continues.
1	main	Higher than threshold (mintthresh.th)	mintstatus.mil	Handler	main	Interrupt level is higher than threshold. Interrupt is taken.

Table 140. Behaviour Between Different Levels of interrupt

Before		Interrupt from CLIC	After			Description
mstatus.mie	pc	Interrupt Level	Interrupt Level Threshold	pc	epc	
0	Handler 1	Don't Care	Don't Care	Handler 1	main	Interrupt is disabled. Handler 1 cannot be pre-empted.
1	Handler 1	0	mintstatus.mil (Handler 1)	Handler 1	main	No interrupt is present. Handler 1 continues.
1	Handler 1	Lower than Handler 1 interrupt level	mintstatus.mil (Handler 1)	Handler 1	main	Interrupt level is lower than threshold. Interrupt is ignored, and Handler 1 continues.
1	Handler 1	Higher than Handler 1 interrupt level	mintstatus.mil (Handler 2)	Handler 2	Handler 1	Interrupt level is higher than threshold. Handler 1 is pre-empted.

4.3.11.2.4. Uninterruptible Critical Section in an Interrupt Handlers

In CLIC mode, an interrupt with lower level is subjected to pre-emption by a higher interrupt level. Any uninterruptible critical section in the interrupt handler with lower level are vulnerable to CLIC's pre-emption, putting the system at risk.

To support uninterruptible critical section in CLIC, Altera recommends either one of the following three software implementations.

Table 141. Software Implementation for Uninterruptible Critical Section

Approach	Description
Non-preemptive	• Synonymous with CLINT. • Without pre-emption, the remaining feature in CLIC is the selective hardware vectoring. All interrupts in CLIC are software vectored because selective hardware vectoring is not supported. • If selective hardware vectoring is not required too, it is recommended to implement CLINT for simplicity, unless more than 16 platform interrupts are required.
Preemptive with global interrupt enable (`mstatus.mie`) disabled	• Utilizes the `mstatus.mie` to temporary disable interrupt. • This prevents any new interrupt from being taken, ensuring the execution of critical section to completion. • Once the critical section is completed, return the `mstatus.mie` back to normal.
Preemptive with interrupt-level threshold (`mintthresh.th`) management	• Temporarily raise `mintthresh.th` to mask a subset of levels. • You may raise the threshold, — Either to the highest level to temporary disable pre-emption, or — A higher level to allow smaller subset of higher interrupt levels to pre-empt. • Once the critical section is completed, return the `mintthresh.th` back to normal.

4.3.11.2.5. Returns from Handlers

When executing an MRET instruction, the processor:

- Returns from a trap,
- Copies the value stored in the `mepc` register to pc, and
- Transfer the `mintstatus.mil` flag to `mcause.mpil`.

4.3.11.3. Software Implementation

The software implementation is not available.

Related Information

Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extensions
For more information about software implementation, refer to the Appendix section.

4.3.12. Memory and I/O Organization

You can configure the Nios V/g processor systems. Consequently, the memory and I/O organization varies from system to system. A Nios V/g processor core uses one or more of the following ports to provide access to memory and I/O:

- Instruction manager port: An Arm Advanced Microcontroller Bus Architecture (AMBA) 4 AXI Memory-Mapped manager port that connects to instruction memory via system interconnect fabric.
- Data manager port: An AMBA 4 AXI Memory-Mapped manager port that connects to data memory and peripherals via the system interconnect fabric.
- Instruction Cache: Fast cache memory internal to the Nios V/g processor core.
- Data Cache: Fast cache memory internal to the Nios V/g processor core.

Nios V/g Processor Core Memory Mapped I/O Access: Both data memory and peripherals are mapped into the address space of the data manager port. Nios V/g processor core uses little-endian byte ordering. Words and half-words are stored in

memory with the more-significant bytes at higher addresses. The Nios V/g processor core does not specify anything about the existence of memory and peripherals. The quantity, type, and connection of memory and peripherals are system dependent.

4.3.12.1. Instruction and Data Buses

4.3.12.1.1. Instruction Manager Port

Nios V/g processor instruction bus is implemented as a 32-bit AMBA 4 AXI manager port.

The instruction manager port:

- Performs a single function: it fetches instructions to be executed by the processor.
- Does not perform any write operations.
- Can issue successive read requests before data return from prior requests.
- Can prefetch sequential instructions.
- Always retrieves 32-bit of data. Every instruction fetch returns a full instruction word, regardless of the width of the target memory. The widths of memory in the Nios V/g processor system is not applicable to the programs. Instruction address is always aligned to a 32-bit word boundary.
- Implements a burst adapter because it is bursting capable and can issue wrapping burst.

Table 142. Instruction Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Unused	[31:0]	Output
	awlen	Unused	[7:0]	Output
	awsize	Unused	[2:0]	Output
	awburst	Unused	[1:0]	Output
	awprot	Unused	[2:0]	Output
	awvalid	Unused	1	Output
	awready	Unused	1	Input
Write Data Channel	wdata	Unused	[31:0]	Output
	wstrb	Unused	[3:0]	Output
	wlast	Unused	1	Output
	wvalid	Unused	1	Output
	wready	Unused	1	Input
Write Response Channel	bresp	Unused	[1:0]	Input
	bvalid	Unused	1	Input
	bready	Unused	1	Output
Read Address Channel	araddr	Instruction Address (Program Counter)	[31:0]	Output

continued...

Interface	Signal	Role	Width	Direction
	arlen	Read burst length • 0 for peripheral region access • 7 for cacheable region access	[7:0]	Output
	arsize	Constant 2 (4 bytes)	[2:0]	Output
	arburst	Constant 2 (WRAP)	[1:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Instruction address valid	1	Output
	arready	Instruction address ready (from memory)	1	Input
Read Data Channel	rdata	Instruction	[31:0]	Input
	rresp	Instruction response: Non-zero value denotes instruction access fault exception	[1:0]	Input
	rlast	Last transfer in a read burst	1	Input
	rvalid	Instruction valid	1	Input
	rready	Constant 1	1	Output

4.3.12.1.2. Data Manager Port

The Nios V/g processor data bus is implemented as a 32-bit AMBA 4 AXI manager port. The data manager port:

- Performs read data from memory or a peripheral when the processor executes a load instruction.
- Performs write data to memory or a peripheral when the processor executes a store instruction.
- Implements a burst adapter because it is bursting capable and can only issue incremental bursts.

axsize signal value indicates the load/store instruction size- byte (LB/SB), halfword (LH/SH) or word (LW/SW). Address on axaddr signal is always aligned to size of the transfer. For store instructions, respective writes strobe bits are asserted to indicate bytes being written.

Nios V/g processor core does not support speculative issue of load/store instructions. Therefore the core can issue only one load or store instruction and it waits until the issued instruction is complete.

Table 143. Data Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Store address	[31:0]	Output
	awlen	Write burst length • 0 for peripheral region access • 7 for cacheable region access	[7:0]	Output
	awsize	Store size: SB, SH, SW	[2:0]	Output
	awprot	Unused	[2:0]	Output

continued...

Interface	Signal	Role	Width	Direction
	awvalid	Store address valid	1	Output
	awready	Store address ready (from memory)	1	Input
Write Data Channel	wdata	Store data	[31:0]	Output
	wstrb	Byte position in word	[3:0]	Output
	wlast	Last transfer in write burst	1	Output
	wvalid	Store data valid	1	Output
	wready	Store data ready (from memory)	1	Input
Write Response Channel	bresp	Store response: Non-zero value denotes store access fault exception.	[1:0]	Input
	bvalid	Store response valid	1	Input
	bready	Constant 1	1	Output
Read Address Channel	araddr	Load address	[31:0]	Output
	arlen	Read burst length • 0 for peripheral region access • 7 for cacheable region access	[7:0]	Output
	arsize	Load size: LB, LH, LW	[2:0]	Output
	arprot	Unused	[2:0]	Output
	arvalid	Load address valid	1	Output
	arready	Load address ready (from subordinates)	1	Input
Read Data Channel	rdata	Load data	[31:0]	Input
	rresp	Load response: Non-zero value denotes load access fault exception	[1:0]	Input
	rlast	Last transfer in a read burst	1	Input
	rvalid	Load data valid	1	Input
	rready	Constant 1	1	Output

4.3.12.2. Address Map

The address map for memories and peripherals in a Nios V/g processor system is design dependent. The following addresses are part of the processor:

1. Reset Address
2. Debug Exception Address
3. Peripheral Region Base Address
4. Exception Address
5. Timer and Software Interrupt Address

You can specify the Reset Address, Debug Exception Address and Peripheral Region Base Address in Platform Designer during system configuration. You can modify the Exception Address stored in the `mvtec` register. `mvtime` and `mtimecmp` register controls the timer interrupt. The `msip` register bit controls the software interrupt.

4.3.12.3. Cache Memory

The Nios V/g processor architecture supports cache memories on both the instruction manager port (instruction cache) and the data manager port (data cache). The cache memories can improve the average memory access time for Nios V/g processor systems that use slow off-chip memory such as SDRAM for program and data storage.

The processor core connects the caches through a 32-bit AXI-4 interface, thus bursting is enabled. The instruction and data caches are always enabled at run-time, but software can bypass the data cache so that peripheral accesses do not return cached data. Software handles cache management and cache coherency. The Nios V/g instruction set provides instructions for cache management.

Table 144. Cache Byte Address Field

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

Related Information

The RISC-V® Instruction Set Manual Volume I: Unprivileged ISA
More information on RISC-V "CMO" Extension for Base Cache Management Operation ISA.

4.3.12.3.1. Instruction Cache

The instruction cache memory has the following characteristics:

- Direct-mapped cache implementation
- 32 bytes (8 words) per cache line
- Configurable size of 0,1, 2, 4, 8, and 16 KBytes
- The instruction manager port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Critical word first
- Burst length of 8

The instruction byte address size is 32-bit. The size of the tag and index field depends only on the size of the cache memory. The offset field is always five bits (i.e., a 32-byte line). The instruction fetch fence (`fence.i`) instruction flushes and invalidates all instruction cache lines.

Related Information

The RISC-V® Instruction Set Manual Volume I: Unprivileged ISA
More information about the the `fence.i` instruction in RV32I Base Integer.

4.3.12.3.2. Data Cache

The data cache memory has the following characteristics:

- Direct-mapped cache implementation
- 32 bytes (8 words) per cache line
- Configurable size of 0,1, 2, 4, 8, and 16 KBytes
- The data manager port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Write-back
- Write-allocate (i.e., on a store instruction, a cache miss allocates the line for that address)
- Burst length of 8

The data byte address size is 32 bit. The size of the `tag` and `index` field depends only on the size of the cache memory. The `offset` field is always five bits (i.e., a 32-byte line).

The Nios V/g processor instruction set provides cache block management instructions for the data cache.

Table 145. RISC-V Zicbom Cache Block Management

Instruction	Name	Operation	Encoding
`cbo.clean <rs1>`[6][7]	Clean Data Cache Address	• Identifies the cache line with `tag` and `index` field. • If there is a cache hit, performs the following operations: — Clear the cache line's dirty state. — If the cache line is valid and dirty, data is written back to the memory.	Refer to *RISC-V Base Cache Management Operation ISA Extension*.
`cbo.flush <rs1>`[6][7]	Flush Data Cache Address	• Identifies the cache line with `tag` and `index` field. • If there is a cache hit, performs the following operations: — Invalidate the cache line. — Write the data back to the memory if the cache line is valid and dirty; otherwise, ignore the data.	
`cbo.inval <rs1>`[6][7]	Invalidate Data Cache Address	• Identifies the cache line with `tag` and `index` field. • If there is a cache hit, performs the following operations: — Invalidate the cache line. — Do not write the data back to the memory.	

(6) Source register 1 (`rs1`) holds the 32-bit cache line address (tag, index, and offset).

(7) If the specified cache line address is not found (cache miss), these instruction are implemented as `nop`, and does no changes to any cache lines.

Table 146. Custom Cache Block Management Instructions

Instruction	Operation	Encoding
`cbo.clean.ix <rs1>`[(8)]	• Identifies the cache line with `index` field, • Clears the cache line's dirty state. • Keeps the cache line's valid state. • If the cache line is valid and dirty, data is written back to the memory.	Refer to Encoding for `cbo.clean.ix`
`cbo.flush.ix <rs1>`[(8)]	• Identifies the cache line with `index` • Invalidates the cache line. • If the cache line is valid and dirty, data is written back to the memory. • If the cache line is not dirty, data is not written back to the memory field,	Refer to Encoding for `cbo.flush.ix`
`cbo.inval.ix <rs1>`[(8)]	• Identifies the cache line with index field, • Invalidates the cache line. • Data is not written back to the memory.	Refer to Encoding for `cbo.inval.ix`

Table 147. Encoding for `cbo.clean.ix`

Bit Field				
31:20	19:15	14:12	11:7	6:0
imm	rs1	cbo	Reserved	misc-mem
000010000001	rs1	010	00000	0001111

Table 148. Encoding for `cbo.flush.ix`

Bit Field				
31:20	19:15	14:12	11:7	6:0
imm	rs1	cbo	Reserved	misc-mem
000010000010	rs1	010	00000	0001111

Table 149. Encoding for `cbo.inval.ix`

Bit Field				
31:20	19:15	14:12	11:7	6:0
imm	rs1	cbo	Reserved	misc-mem
000010000000	rs1	010	00000	0001111

4.3.12.3.3. Configurable Cache Memory

You can configure the size of the cache memory. The cache memory has no effect on program functionality but affects the speed with which the processor fetches instructions and reads/writes data.

(8) Source register 1 (`rs1`) holds the cache line index, which can be 5 to 9-bits depending on the cache size.

4.3.12.3.4. Bypassing Cache (Peripheral Region)

The Nios V/g architecture has two peripheral regions for bypassing the caches. Nios V/g cores optionally support the peripheral region mechanism to indicate cacheability. In the Platform Designer, the peripheral region cache-ability mechanism allows you to specify a region of address space that is non-cacheable. The peripheral region is any integer power of 2 bytes, from a minimum of 64 kilobytes up to a maximum of 2 gigabytes, and must be located at a base address aligned to the size of the peripheral region. The peripheral region is available provided that an MMU is not present. The peripheral regions are non-bursting.

Note: Any accesses to the Nios V processor's debug module or timer module are non-cacheable.

Note: You must place the peripherals driven by the Nios V/g processor within a defined peripheral region to achieve cache bypass, which is required by a standard design implementation.

4.3.12.3.5. Using Cache Memory Effectively

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

- Regular memory is located off-chip and has a longer access time than 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.

The optimal cache configuration is application-specific, but you can define a configuration that works for a variety of applications. Refer to the following examples:

- If a Nios V/g processor system only has fast on-chip memory and never accesses slow off-chip memory, an instruction or data cache is unlikely to boost the performance.
- If a program's critical loop is 2 KB but the instruction cache is 1 KB, an instruction cache does not improve execution speed. In this case, an instruction cache can actually degrade performance.

4.3.12.4. Tightly Coupled Memory

Tightly coupled memory (TCM) guarantees fixed low-latency memory access for performance-critical applications. The advantages of TCM over cache memory are as follows:

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

Physically, a TCM is a dedicated on-chip memory within the Nios V processor core. Altera implements TCM using the M20K memory blocks. The Nios V processor architecture supports four TCMs for instruction access and data access (two per access type). Each TCM provide an AXI4-Lite interface for connection with an external master, or manager. The interconnect allows you to connect Avalon or other supported manager, gaining access to read or write the respective TCMs.

4.3.12.4.1. Instruction and Data Tightly Coupled Memory

Nios V/g processor supports the following TCMs:

- Two instruction TCMs
- Two data TCMs

Table 150. TCM Characteristics

Characteristics	TCM
4 bytes (1 words) per address line	• Instruction TCM • Data TCM
Fixed memory latency of 1 cycle	• Instruction TCM • Data TCM
Configurable size of 0 (Disabled) to 512MBytes	• Instruction TCM • Data TCM
Configurable 32-bits base address	• Instruction TCM • Data TCM
Supports memory initialization using MIF or HEX file	• Instruction TCM • Data TCM
Read-only permission for processor core	Instruction TCM
Read/Write permission for external AXI4-Lite manager	Instruction TCM
Read/Write permission for processor core and external AXI4-Lite manager	Data TCM

Table 151. Tightly Coupled Memory Interface Signals

Interface	Signal	Role	Width	Direction
Write Address Channel	awaddr	Write address	Width = log (tcm_size)/log 2[9]	Input
	awprot	Unused	[2:0]	Input
	awvalid	Write address valid	1	Input
	awready	Write address ready (from TCM)	1	Output
Write Data Channel	wvalid	Write data valid	1	Input
	wdata	Write data	[31:0]	Input
	wstrb	Byte position in word	[3:0]	Input
	wready	Write data ready (from TCM)	1	Output
Write Response Channel	bvalid	Write response valid	1	Output
	bresp	Write response: Non-zero value denotes store access fault exception	[1:0]	Output
	bready	Write response ready (from external manager)	1	Input
Read Address Channel	araddr	Read address	Width = log (tcm_size)/log 2[9]	Input

continued...

(9) The width of the Write and Read Addresses depends on the selected TCM size.

Interface	Signal	Role	Width	Direction
	arprot	Unused	[2:0]	Input
	arvalid	Read address valid	1	Input
	arready	Read address ready (from TCM)	1	Output
Read Data Channel	rdata	Read data	[31:0]	Output
	rvalid	Read data valid	1	Output
	rresp	Read data response: Non-zero value denotes load access fault exception	[1:0]	Output
	rready	Read data ready (from TCM)	1	Input

4.3.12.4.2. Accessing Tightly Coupled Memory

TCM occupies standard address space like other memory devices connected via system interconnect fabric. The processor configures the address ranges for TCM (if any).

Access from Processor Core

When TCM is present, the Nios V/g processor core decodes addresses internally to determine if the requested addresses reside in the TCMs. If the address resides in the TCM, the processor core fetches the instruction from instruction TCM or loads the data from the data TCM. The software accesses TCM using regular load and store instructions. From the software's perspective, there is no difference in accessing a TCM compared to other memory.

Accessing TCM bypasses cache memory. The processor core functions as if the cache were not present for the address span of the TCM. Instructions for managing the cache do not affect the TCM, even if the instruction specifies an address in TCM.

Access from External AXI4-Lite Manager

Any external AXI4-Lite manager can access any TCMs as RAM in the system if it is connected to the TCM's AXI4-Lite interface. The TCMs support dual-port access, which allows two hosts (processor core and external AXI4-Lite manager) to access the memory simultaneously.

You can access the TCM memories of a Nios V processor core (Core 1) from another core (Core 2) in a multi-core system. Core 2 acts as the external AXI4-Lite Manager in this setup. Enable this feature by connecting the data bus of Core 2 to the TCM AXI4-Lite subordinate port of Core 1.

4.3.12.4.3. Using Tightly Coupled Memory Effectively

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

You can implement one or more of the following functions or modules using TCMs to enhance the performance of your system:

- Constant access time with no arbitration delays.
- Separate exception stack for use only while handling interrupts.
- Fast data buffers
- Fast sections of code:
  - Fast interrupt handler
  - Critical loop

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.

Consider the following design when using TCMs:

- Strike a balance between the speed enhancement from caches with the higher speed enhancement from TCMs.
- Divide the on-chip memory resources equitably for the best combination of TCMs and cache.
- Locating any information (code or data) within TCMs eliminates cache overhead such as cache flushing, loading, or invalidating.

Note: You can connect Nios V processor data manager to instruction TCM subordinate port. Like Nios II processor, the Instruction TCM can map into the processor's data region for memory debugging.

Related Information

AN 978: Nios® V Processor Migration Guidelines
Refer to the Nios V V Processor Migration Guide for the difference between Nios V andNios II processors TCMs.

4.3.13. RISC-V based Debug Module

The Nios V/g processor architecture supports a RISC-V based 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 debug module and provide facilities, such as the following features:

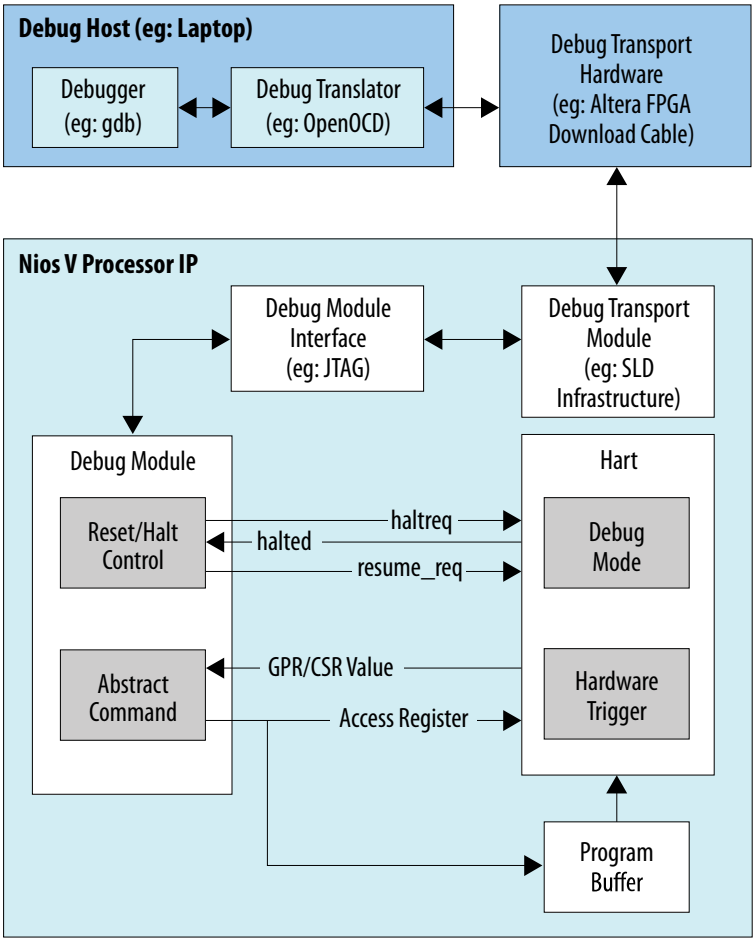
- Reset Nios V processor core and timer module
- Download programs to memory
- Start and stop execution
- Set software breakpoints and watchpoints
- Analyze registers and memory

You can interact with the Debug Host (e.g., laptop), which is running a debugger (e.g., gdb). The debugger communicates with a Debug Translator (e.g., OpenOCD) to communicate with Debug Transport Hardware (e.g., Altera FPGA Download Cable). The Debug Transport Hardware connects the Debug Host to the Nios V processor's Debug Transport Module (DTM). The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

Every Nios V processor has a single hart, which is controlled by a single DM. DMs provide reset/halt control of the Nios V processor. Abstract commands provide access to GPRs. Additional registers are accessible through abstract commands or the Program Buffer. In addition to register access, the Program Buffer also allows the debugger to execute arbitrary instructions and access memory.

The Nios V processor implements a Trigger Module. When trigger conditions are met, the processor halts and informs the debug module that it has halted.

Figure 19. Nios V Processor Debug System Overview



4.3.13.1. Debug Module

The Debug Module supports the following operations:

1. Provide information about the processor's implementation to the debugger.
2. Allow users to halt and resume the processor.
3. Provide halt status on the processor.
4. Provide abstract read and write access to the halted processor's GPRs through abstract command.

5. Provide `ndm_reset_in` and `dbg_reset_out` signals that allow debugging from the reset vector after reset.
6. Provide abstract access to non-GPR hart registers through abstract command.
7. Provide a Program Buffer to force the hart to execute arbitrary instructions through abstract commands.

The Nios V processor supports only one hart. Therefore:

- You cannot apply hart selection.
- The bit fields that differentiate between "all harts" and "any harts" are treated as the same.
- It simplifies related bit fields(such as `dmstatus.allhavereset` and `dmstatus.anyhavereset` to `dmstatus.*havereset` throughout the document.

4.3.13.1.1. Reset/Halt/Run Control

In the Nios V processor, the Debug Module tracks 3 conceptual bits of state:

- halt request
- resume ack
- hart reset

In the `dmstatus` register, the debugger can observe the state of resume ack in `dmstatus.*resumeack`, and the state of halted, running, and havereset signals in `dmstatus.*halted`, `dmstatus.*running`, and `dmstatus.*havereset`.

Reset processor

Debug Module controls a global reset signal, `dmcontrol.ndmreset` (non-debug module reset), which can reset, or hold in reset, every component in the platform, except for the Debug Module and Debug Transport Modules. The debugger can reset the processor by asserting `dmcontrol.ndmreset`, through the `ndm_reset_in` and `dbg_reset_out` signals. Refer to Reset and Debug Signals about how to connect the reset signals. Next, check the reset status using `dmstatus.*havereset` bit field.

Halt processor

When the processor comes out of reset and `dmcontrol.haltreq` are set, the hart immediately halt and enter Debug Mode (halted state). Next, check the halted status using `dmstatus.*halted` bit field.

Resume processor

To resume the processor's execution, debugger writes 1 to `dmcontrol.resumereq`. Next, check the resume status using `dmstatus.*resumeack` bit field. After that, `dmstatus.*running` is set to 1 indicating the processor is running in normal execution.

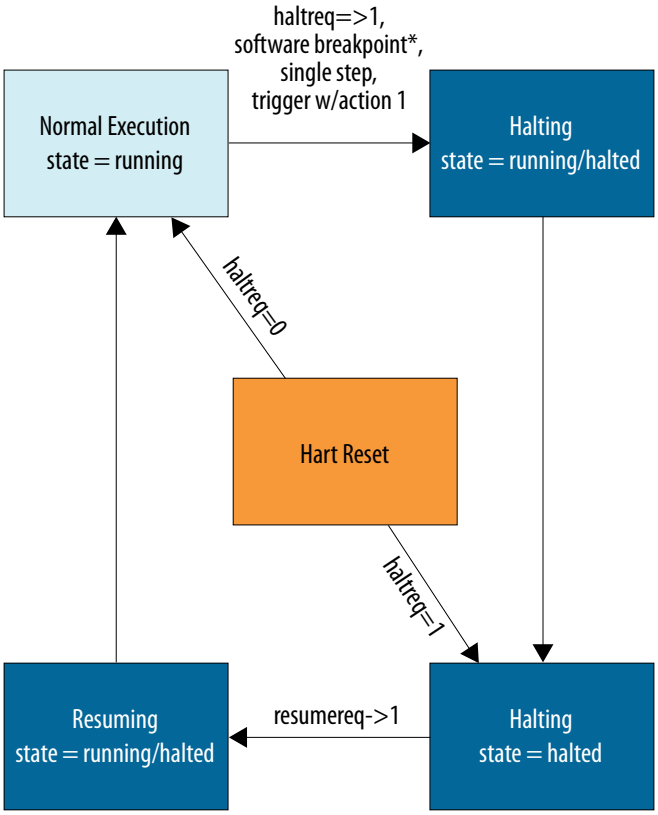
4.3.13.1.2. Hart States

The processor is under one of the following four states at any given time:

- Non-existent (Probing other harts which does not exist)
- Unavailable (Processor being reset)
- Running (Processor executing normally outside of debug)
- Halted (Processor in Debug Mode)

They are reflected in the `dmstatus` register, as `dmstatus.*nonexistent`, `dmstatus.*unavail`, `dmstatus.*running` and `dmstatus.*halted` respectively.

Figure 20. Relationship between Hart DM States



4.3.13.1.3. Debug Module Register

The Debug Module registers are accessible over the DMI bus only. Each register has a fixed address and developed based on the RISC-V Debug Specification 0.13.2, with JTAG DTM. The Debugger can determine the register implementation status by writing to or reading from the Debug Module registers. Unimplemented registers return 0 when read.

Table 152. Debug Module Register

Address	Name	Description
0x04	Abstract Data 0 (`data0`)	Refer to Abstract Commands.
0x05	Abstract Data 1 (`data1`)	Refer to Abstract Commands.
0x10	Debug Module Control (`dmcontrol`)	• There is a single hart selected. • `hartsel` is 0.
0x11	Debug Module Status (`dmstatus`)	• Supports implicit ebreak instruction after Program Buffer execution. Authentication is not implemented. `authenticated` and `authbusy` hardwired to 1. • `confstrptr` is not relevant to configuration string. • Halt-on-reset functionality is not supported. • Version = 3
0x12	Hart Info (`hartinfo`)	• No `dscratch` registers available during Program Buffer execution. • `data0` holds arg0/return value • `data1` holds arg1 • No CSRs dedicated to shadowing the `data` registers.
0x14	Hart Array Window Select (`hawindowsel`)	Unimplemented register. Nios V processor contains a single hart only.
0x15	Hart Array Window (`hawindow`)	Unimplemented register. Nios V processor contains a single hart only.
0x16	Abstract Control and Status (abstractcs)	• Eight (8) Program Buffer, in 32-bits words. • Two (2) `data` registers Refer to Abstract Commands.
0x17	Abstract Command (command)	Refer to Abstract Commands.
0x18	Abstract Command Autoexec (abstractauto)	Unimplemented register.
0x19 ~ 0x1c	Configuration String Pointers (confstrptr)	Unimplemented registers.
0x1d	Next Debug Module (nextdm)	Unimplemented register. There is only a single DM.
0x20 ~ 0x27	Program Buffer 0 to 7	Refer to Program Buffer.
0x30	Authentication Data	0x0
0x37 ~ 0x3f	System Bus Registers	Unimplemented registers. System Bus is not supported.
0x40	Halt Summary 0	LSB reflects halt status of the Nios V processor.

4.3.13.2. Abstract Commands

Nios V processor DM supports Access Register Command only (`command.cmdtype` = 0). You can determine that only Access Register abstract commands is supported by attempting them and then check the abstractcs.cmderr. Other abstract commands are not supported (`abstractcs.cmderr` = 2).

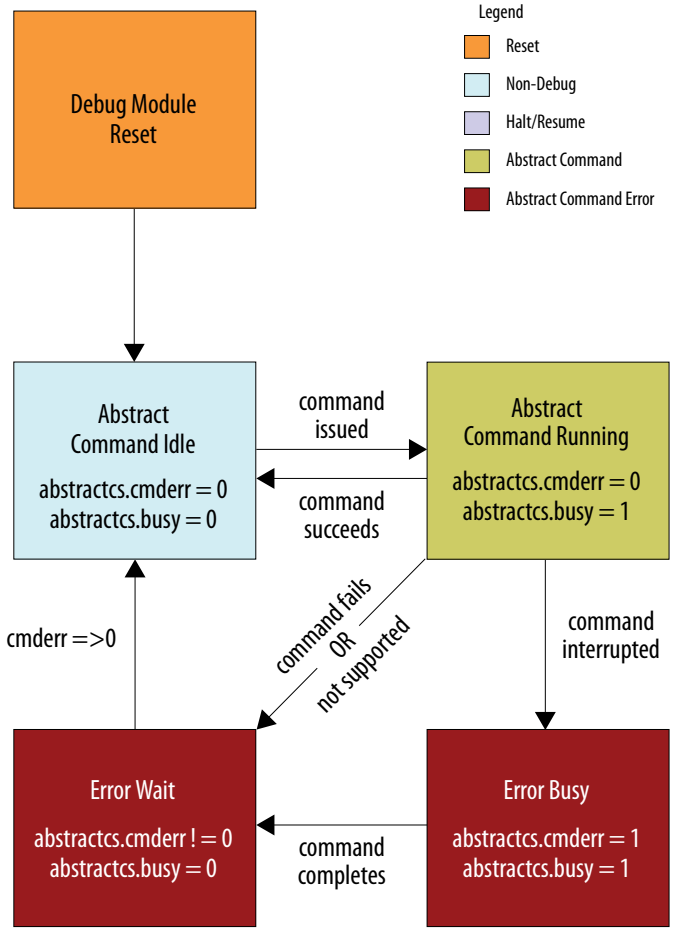
Debuggers execute abstract commands by writing them to command and check its completion by reading `abstractcs.busy`. If the debugger starts a new command while abstractcs.busy is set, `abstractcs.cmderr` becomes 1 (busy), the currently

executing command continues without interruption, but any error generated from its execution is lost. After completion, `abstractcs.cmderr` indicates whether the command was successful. Commands may fail because the processor is not halted, not running, unavailable, or encounter an error during execution.

If the command takes arguments, the debugger must write them to the data registers before writing to `command`. If a command returns results, the results are placed in the `data` registers before `abstractcs.busy` is cleared.

Note: The Nios V processor does not support abstract commands when hardware thread is not halted.

Figure 21. Behaviour of Abstract Commands



4.3.13.2.1. Access Register Command

The Access Register command allows read-write access to the processor registers including GPRs, CSRs, FP registers and Program Counter (dpc in Debug Mode). The Access Register also allows program execution from program buffer. The debugger executes Access Register commands by writing into `command` register using the Access Register command encoding.

Sequence to execute Access Register Command:

1. If `command.write` is 0 and `command.transfer` is 1, then read data from the register specified by command.regno into the data0.
2. If `command.write` is 1 and `command.transfer` is 1, then write data from the data0 into the register specified by `command.regno`.
3. If `command.aarpostincrement` is 1, increment `command.regno`.
4. If `command.postexec` is 1, execute the Program Buffer.

Figure 22. Accessing Access Register Command

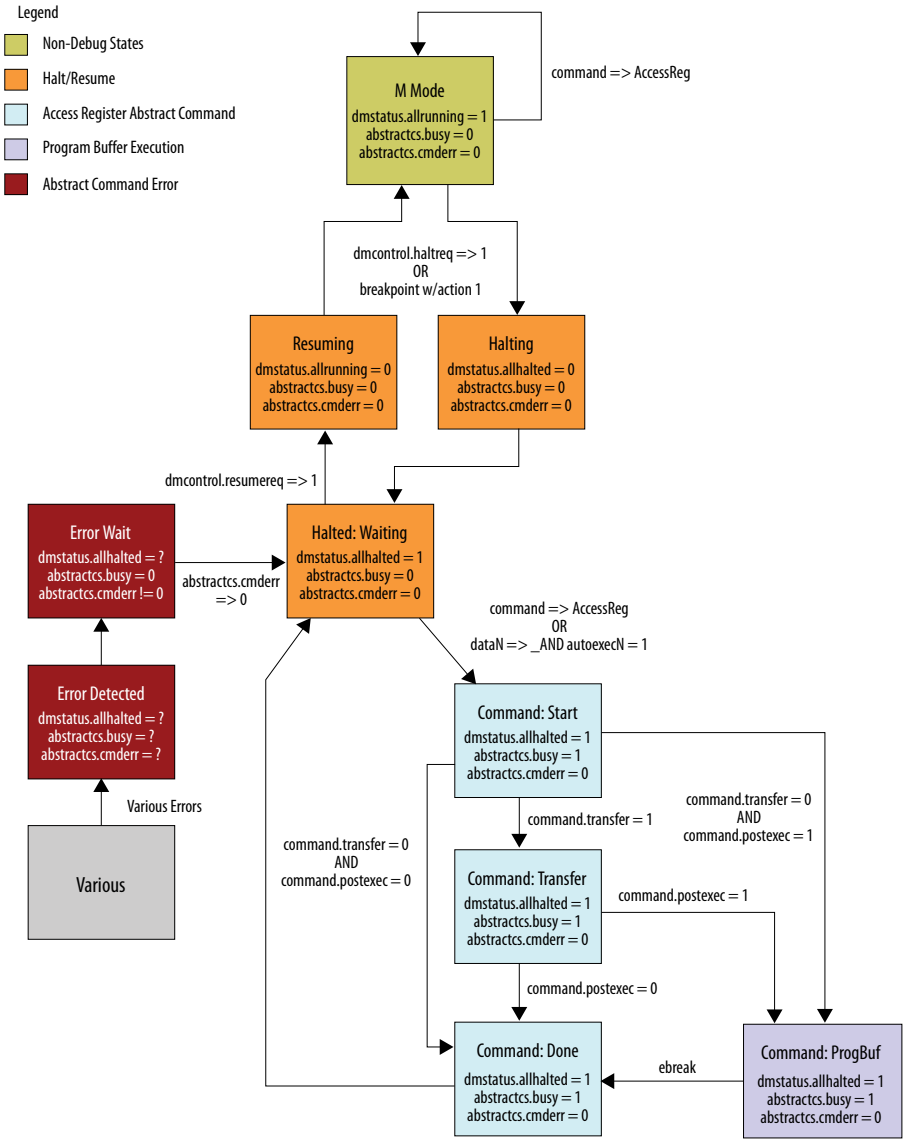


Table 153. Access Register Command Encoding

Bit Field							
31 - 24	23	22 - 20	19	18	17	16	15-0
cmdtype=0	0	aarsize=2	aarpostincrement	postexec	transfer	write	regno

Table 154. Field Description

Field	Description
cmdtype	0 indicates Access Register command.
aarsize	2 indicates 32-bit register access. Others are not applicable in 32-bit Nios V processor.
aarpostincrement	0: No effect 1: Increment regno after successful register access
postexec	0: No effect 1: Execute program in program buffer
transfer	Acts in conjunction with write field. 0: Ignore value in write field 1: Execute operation specified by write field.
write	0: Copy data from register (Read) 1: Copy data to register (Write)
regno	Register address to be accessed • 0x0 ~ 0xfff : CSRs (Refer to the topic Control and Status Registers (CSR) Mapping (CLINT) for available CSRs). • 0x1000 ~ 0x101f : GPRs • 0x1020 ~ 0x103f : Floating Point Register

If the Access Register command fails, the DM stores root causes in the `abstractcs.cmderr` field. The `abstractcs.cmderr` field represents the current state of abstract command being executed. If the `abstractcs.cmderr` field is non-zero, it ignores the writes to the command register. To clear the `abstractcs.cmderr` field, write 1 for every bit in the field.

Table 155. cmderr Description

cmderr	Description
0	No error.
1	DM is busy.
2	The command is not supported.
3	An exception occurred when executing the command.
4	The processor is not halted.
5	Experiencing bus error.
7	Others

For more information about Access Register command, refer to The RISC-V Debug Specification.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

4.3.13.2.2. Using Access Register Command

Read `s0` using abstract command:

Op	Address	Value	Comment
Write	`command`	`aarsize=2, transfer, regno= 0x1008`	Read s0
Read	`data0`	—	Returns value that was in s0

Write `mstatus` using abstract command:

Op	Address	Value	Comment
Write	`data0`	new value	—
Read	`command`	`aarsize=2, transfer, regno= 0x300`	Write `mstatus`

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

4.3.13.3. Program Buffer

Debugger can write small programs into Program Buffer to execute arbitrary instruction, then use Access Register command to initiate the execution (setting `command.postexec`).

Note that the Nios V processor Debug Module supports implied ebreak at the non-existent word immediately after the Program Buffer. This saves the debugger from having to write the ebreak itself, and allows the sequence of instructions to be one word smaller.

4.3.13.3.1. Using Program Buffer

Table 156. Write `mstatus` using Program Buffer

Op	Address	Value	Comment
Write	`progbuf0`	`csrw s0, MSTATUS`	—
Write	`progbuf1`	`ebreak`	—
Write	`data0`	`new value`	—
Write	`command`	`aarsize=2, postexec, transfer, write, regno= 0x1008`	Write `s0`, then excecute program buffer.

Table 157. Read `f1` using Program Buffer

Op	Address	Value	Comment
Write	`progbuf0`	`fmv.x.s s0, f1`	—
Write	`progbuf1`	`ebreak`	—

continued...

Op	Address	Value	Comment
Write	command	postexec	Execute program buffer
Write	command	transfer, regno=0x1008	read s0
Read	data0	—	Returns the value that was in f1

Table 158. Read a Word from Memory using Program Buffer

Op	Address	Value	Comment
Write	progbuf0	lw S0, 0(s0)	—
Write	progbuf1	ebreak	—
Write	data0	address	—
Write	command	write, postexec, regno=0x1008	Write s0, then execute program buffer
Write	command	regno=0x1008	Read s0
Read	data0	—	Value read from memory

Table 159. Write a Word to Memory Using Program Buffer

Op	Address	Value	Comment
Write	progbuf0	sw s1, 0(s0)	—
Write	progbuf1	ebreak	
Write	data0	address	—
Write	command	write, regno=0x1008	Write s0
Write	data0	value	—
Write	command	write, postexec, regno=0x1009	Write s1, then execute program buffer.

4.3.13.4. Debug Mode

4.3.13.4.1. Enter Debug Mode

You can enter the Debug Mode, as specified in the RISC-V architecture specification, in the following ways:

1. Halt from Debug Module
2. Software breakpoints
3. Trigger or Hardware breakpoint

Upon entering Debug Mode, Nios V processor completes the instruction in W-stage. By the order of priority, instruction in M-stage, E-stage, D-stage or F-stage takes the interrupt.

- When there is a valid instruction, Program Counter writes to the Debug Program Counter, `.dpc`.
- If the instruction in M-stage is not valid, then instruction in E-stage takes the interrupt and so on and so forth.
- If there is no valid instruction in the pipeline, the Program Counter for the next instruction writes to the Debug Program Counter, `.dpc`.

Note: For branches, the next Program Counter depends on whether a branch was taken or not taken, and whether the branch prediction (if any) was correct.

4.3.13.4.2. Exit Debug Mode

Executing the dret instruction in Debug Mode changes pc to the value stored in dpc. The processor then enters Machine Mode and exits Debug Mode.

4.3.13.4.3. Core Debug Registers

These registers are only accessible from Debug Mode, using the RISC-V csr opcodes or abstract debug commands.

Table 160. Core Debug Registers

Name	Name	Description
dcsr	Debug Control and Status	• ebreak instructions in M-mode enter Debug Mode. • Interrupts are disabled during single stepping. • Increment counters and timers as usual. • MPRV in mstatus is ignored in Debug Mode.
dpc	Debug PC	—
dscratch0	Debug Scratch Register 0	—

4.3.13.5. Hardware Trigger Module

The Nios V processor core supports one address or data match trigger. The trigger registers are accessible using RISC-V csr opcodes or abstract debug commands. The firing of the trigger can either enter the Debug Mode or raise a breakpoint exception, which depends on the trigger registers.

Without applying software breakpoint, this makes hardware triggers invaluable when debugging code from ROM.

Note: The trigger is disabled in the following situations:

- The Nios V processor is in debug mode.
- The Nios V processor is in machine mode, while `mcontrol.m` or `mcontrol.type` is 0.

4.3.13.5.1. Trigger Registers

Table 161. Trigger Registers Implemented in Nios V Processor

Name	Registers	Description
`tselect`	Trigger Select	Nios V processor supports one trigger, therefore the processor selects Trigger 0 at default (value set at 0).
`tdata1`	Trigger Data 1	The value of type field is 2 to represent Trigger 0 as an address or data match trigger. Write behavior to tdata registers depends on the dmode filed. The remaining bits acts as `mcontrol`.
`mcontrol`	Match Control	Controls address and data trigger implementation according to `action`, `m`, `execute`, `store`, and `load` fields.
`tdata2`	Trigger Data 2	Holds trigger-spesific data (virtual address, instruction opcode, data stored or loaded).
`tinfo`	Trigger Info	The value of info field is 4 at default to signify `tdata1.type` is 2. This implies selected trigger is and address or data match trigger.

4.3.13.5.2. Type of Address/Data Match Triggers

Base on the bit field setting in `mcontrol` register, Nios V processor can implement different trigger types after the selected trigger matches the tdata2 register.

You can configure the hardware triggers to fire (through `mcontrol` register), when matches:

- Instruction address
- Instruction opcode
- Store address
- Store data
- Load address
- Load data

Table 162. Address or Data Match Trigger Type

Trigger	`mcontrol` bit fields			
	`select`	`execute`	`store`	`load`
Instruction Address	0	1	0	0
Instruction Opcode	1	1	0	0
Store Address	0	0	1	0
Store Data	1	0	1	0
Load Address	0	0	0	1
Load Data	1	0	0	1

Based on the selected trigger, the trigger firing time and the behaviour of the `mepc` register can varies.

Table 163. Nios V Processor Triggers Condition and Firing Time

Triggers	Condition	Firing Time	Exception Program Counter
Instruction Address Trigger	Program Counter matches `tdata2`	Before executing the instruction	The processor sets the Machine Exception Program Counter (`mepc`) to the instruction address (PC).
Instruction Opcode Trigger	Instruction opcode matches `tdata2`	Before executing the instruction	
Store Address Trigger	Store address matches `tdata2`	After executing the store instruction	The processor sets the `mepc` to the next instruction address (PC + 4).
Store Data Trigger	Store data matches `tdata2`	After executing the store instruction	
Load Address Trigger	Load address matches `tdata2`	After executing the load instruction	
Load Data Trigger	Load data matches `tdata2`	After executing the load instruction	

4.3.13.5.3. Using Triggers

A debugger can use hardware triggers to halt a hart (or enter Debug Mode) when a certain event occurs. In Nios V processor, the supported trigger type is address/data match trigger. Thus, the associated events are related to address and data trigger implementation only. When a debugger wants to set a trigger, the debugger writes the desired configuration, and then reads back to see if that configuration is supported.

Table 164. Using Triggers Example 1

Enter Debug Mode just before the instruction at 0x8001234 is executed. Use as an instruction breakpoint in ROM.

Name	Address	Description
`tdata1`	0x105c	action=1, match=0, m=1, s=1, u=1, execute=1
`tdata2`	0x80001234	address

Table 165. Using Triggers Example 2

Enter Debug Mode right after the value at 0x80007f80 is read.

Name	Address	Description
`tdata1`	0x4159	timing-1, action=1, match=0, m=1, s=1, u=1, load=1
`tdata2`	0x80007f80	address

Table 166. Using Triggers Example 3

Enter Debug Mode right before a write to an address between 0x81230007c80 and 0x80007cef (inclusive).

Name	Address	Description
`tdata1 0`	0x195a	action=1, chain=1, match=2, m=1, s=1, u=1, store=1
`tdata2 0`	0x80007c80	start address (inclusive)
`tadata1 1`	0x11da	action=1, match=3, m=1, s=1, u=1, store=1
`tdata2 1`	0x80007cf0	end address (exclusive)

Table 167. Using Triggers Example 4

Enter Debug Mode right before a write to an address between 0x8123000 and 0x8123ffff.

Name	Address	Description
tdata1	0x10da	action=1. match=1, m=1, s=1, u=1. store=1
tdata2	0x81237fff	16 bits to match exactly, then 0, then all ones

Table 168. Using Triggers Example 5

Enter Debug Mode right after a read from an address between 0x86753090 and 0x8675309f or between 0x96753090 and 0x96753309f (inclusive).

Name	Address	Description
tdata1 0	0x41a59	timing=1, action=1, chain=1, match=4, m=1, s=1. u=1, load=1
tdata2 0	0xfff03090	Mask for low half, then match for low half
tadata1 1	0x412d9	timing=1, action=1, match=5, m=1, s=1, u=1, load=1
tdata2 1	0xefff8675	Mask for high half, then match for high half

4.3.13.6. JTAG Debug Transport Module

The JTAG DTM provides access to the DM over JTAG, developed based on RISC-V Debug Specification 0.13.2, with JTAG DTM.

4.3.13.6.1. JTAG DTM Registers

When the JTAG TAP is reset, IR is 00001 at default (selecting the IDCODE instruction).

Table 169. JTAG DM Register

Address	Name	Description
0x00	BYPASS	JTAG recommends this encoding
0x01	IDCODE	JTAG recommends this encoding
0x10	DTM Control and Status (dtmcs)	Refer to .
0x11	Debug Module Interface Access (dmi)	Refer to *RISC-V Debug Specification 0.13.2, with JTAG DTM*.
0x1f	BYPASS	JTAG requires this encoding

4.3.13.6.2. Accessing Debug Module Interface

Altera recommends you to utilize the OpenOCD executables along with the RISC-V Debug Configuration, Authentication, and DMI commands to access the Debug Module. They offer abstraction to the JTAG DTM, which reduces complexity and improve efficiency during Debug Module access.

If you are developing a custom Debug Translator, please refer to subchapter *B.1 Debug Module Interface Access* in *RISC-V Debug Specification 0.13.2, with JTAG DTM*.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

4.3.13.7. Nios V Processor Implementation

The Nios V processor implements the Execution-Based. This implementation only implements the Access Register abstract command for GPRs on a halted hart, and relies on the Program Buffer for all other operations. It uses the hart's existing pipeline and ability to execute from arbitrary memory locations to avoid modifications to a hart's datapath.

For more information about Execution Based implementation, refer to *RISC-V Debug Specification 0.13.2, with JTAG DTM* in Appendix *Execution Based*.

Related Information

RISC-V Debug Specification 0.13.2, with JTAG DTM

4.3.14. Error Correction Code (ECC)

The Nios V/g processor core has the option to enable error detection and ECC status for the following internal RAM blocks:

- Register file
- Instruction cache
- Data cache
- Tightly coupled memories

Each RAM block has its source ID. When an ECC event occurs, the processor transmits the source ID and ECC status to the ECC interface.

Table 170. ECC Implementation

Single Bit Correction	ECC Implementation
Disabled	• If the ECC event is a correctable error, the processor continues to operate without correcting the error. • If the ECC event is an un-correctable error, the processor halts its current progress and stall. You need to reset either the processor core alone, or entire system.
Enabled	• If the ECC event is a correctable error, the processor continues to operate, and corrects the memory source. • If the ECC event is an un-correctable error, the processor halts its current progress and stall. You need to reset either the processor core alone, or entire system.

Note: To reset the processor core alone, you need to apply the **Reset Request Interface** to safely reset the Nios V processor (Cleared of any outstanding operations). To reset the entire system, you can use the hard reset interface instead.

Related Information

Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide

4.3.14.1. ECC Interface

The ECC interface allows external logic to monitor ECC errors in the Nios V/g processor. The interface is a conduit and made up of two output signals.

- `cpu_ecc_status`: Indicates the error status.
- `cpu_ecc_source`: Indicates the error source

Table 171. `cpu_ecc_status`

2-bits Encoding	ECC Status	Effects on Software	Next Action
2'b00	No ECC event	None	None
2'b01	Reserved	Not Applicable	Not Applicable
2'b10	Correctable single bit ECC error	None	None
2'b11	Un-correctable ECC error	Likely fatal and halts processor	Reset either processor or entire system

Table 172. `cpu_ecc_source`

4-bits Encoding	ECC Source	Available
4'b0000	No ECC event	Always
4'b0001	General Purpose Register (GPR)	Always
4'b0010	Instruction Cache Data RAM	Always
4'b0011	Instruction Cache Tag RAM	Always
4'b0100	Data Cache Data RAM	Always
4'b0101	Data Cache Tag RAM	Always
4'b0110	Instruction TCM1	When Instruction TCM1 is present
4'b0111	Instruction TCM2	When Instruction TCM2 is present
4'b1000	Data TCM1	When Data TCM1 is present
4'b1001	Data TCM2	When Data TCM2 is present
4'b1010	Floating Point Register (FPR)	When FPU is present
4'b1111	Reserved	Not Applicable

4.3.14.2. ECC Error Injection

ECC error injection allows you to inject ECC errors into the processor. This ECC error injection tests the hardware features, firmware, and driver development. You can trigger an ECC exception by writing an error code into the ECC Error Inject module based on the `mtval2` table.

Note: To prevent the hardware from continuing to trigger the ECC error, write a zero to turn off the ECC error injection.

Altera provides an error injection API for users to inject an ECC error into the processor based on the Table *List of ECC Sources based on `mtval2` Bit Field*. The source header file is in `<BSP Project>\HAL\inc\sys\alt_ecc_error_inject.h`.

```
alt_ecc_error_inject(ALT_ECC_ERROR_TYPE type)I
```

For example, if you write 24 to the module, the `cpu_ecc_status` and `cpu_ecc_source` of Data TCM1 become 2'b10 and 4'b1000 respectively. If you write 25 to the module, the `cpu_ecc_status` and `cpu_ecc_source` of Data TCM1 become 2'b11 and 4'b1000, respectively.

Table 173. How ECC Error Injection Works

ECC Error Inject	Module based on mtval2	`cpu_ecc_status`	`cpu_ecc_source`
24	Data TCM1 Correctable Error	2'b10 (Correctable)	4'b1000 (Data TCM1)
25	Data TCM1 Uncorrectable Error	2'b11 (Uncorrectable)	4'b1000 (Data TCM1)

4.3.14.3. Affected CSR during ECC Event

The CSRs are involved in the ECC: `mcause`, `mtval` and `mtval2`. For more information, refer to the *Control and Status Register Field*.

Table 174. List of Affected CSR

Affected CSR	Description
`mcause`	An exception code 19 (0x13 – Hardware Error Exception) is written to the mcause.
`mtval`	Contains the violating address when an ECC error occurs during a load/store or instruction fetch error.
`mtval2`	Provides more information for an ECC event. Refer to the table below.

Table 175. List of ECC Sources based on `mtval2` Bit Field

mtval2 Value	ECC Error Source
0 (32'h0)	No Error
1 (32'h1)	GPR ECC Uncorrectable Error
3 (32'h3)	FPR ECC Uncorrectable Error
16 (32'h10)	Instruction TCM1 Correctable Error
17 (32'h11)	Instruction TCM1 Uncorrectable Error
18 (32'h12)	Instruction TCM2 Correctable Error
19 (32'h13)	Instruction TCM2 Uncorrectable Error
24 (32'h18)	Data TCM1 Correctable Error
25 (32'h19)	Data TCM1 Uncorrectable Error
26 (32'h1A)	Data TCM2 Correctable Error
27 (32'h1B)	Data TMC2 Uncorrectable Error
33 (32'h21)	Instruction Cache TAG RAM Uncorrectable Error
35 (32'h23)	Instruction Cache Data RAM Uncorrectable Error
41 (32'h29)	Data Cache TAG RAM Uncorrectable Error
43 (32'h2B)	Data Cache Data RAM Uncorrectable Error
Others	Reserved

4.3.15. Branch Prediction

Nios V/g processor core supports Static Branch Prediction. This core implements Backward Taken, Forward Not Taken (BTFN) mechanism. This technique is a simple form of branch prediction because it does not rely on the history of the branches. It predicts the outcome of a branch instruction as follows:

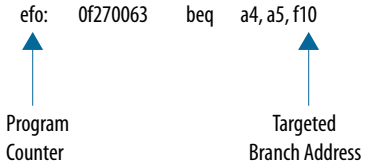
- Always taken, when backward branches, or
- Always not taken when forward branches.

The processor core determines the type of branches by comparing the Program Counter and targeted branch address (branch PC).

- Backward branch - When the target branch address is less than the Program Counter.
- Forward branch - When the target branch address is greater than the Program Counter.

For example, the assembly code below shows a forward branch. ef0 is the Program Counter, and f10 is the targeted branch address. Since the targeted branch address is greater than the Program Counter, the processor core does not take the branch.

Figure 23. Branch Prediction



Note: Enabling branch prediction can improve the core performance (such as the Dhrystone and CoreMark benchmark), but it may also increase logic resource utilization.

4.3.16. Lockstep Module

The Nios V Processor Lockstep feature utilizes fRSmartComp technology to implement a smart comparator in RTL. Altera utilizes the Dual-Core Lock Step (DCLS) safety architecture to implement the smart comparator and integrates fRSmartComp technology into the Nios V/g processor, allowing for the design of fail-safe applications.

Related Information

Nios V Processor: Lockstep Implementation

4.4. Programming Model

4.4.1. Privilege Levels

The privilege levels in Nios V/g processor are designed based on the RISC-V architecture specification. The privilege levels available are Machine Mode (M-mode) and Debug Mode (D-mode).

Related Information

RISC-V® Specifications

4.4.1.1. Machine Mode (M-mode)

The default operating mode is Machine mode (M-mode). The core boots up into M-mode after power-on reset. M-mode provides low-level access to the machine implementation and all CSRs.

4.4.1.2. Debug Mode (D-mode)

The Debug mode (D-mode) is an additional privilege level to support off-chip debugging and manufacturing test.

The core can enter D-mode using any of the following methods:

- After reset when bit is set in debug CSR.
- Using debug interrupt.
- Using EBREAK instruction when bit is set in debug CSR.
- Using single step.

4.4.2. Control and Status Registers (CSR) Mapping (CLINT)

Control and status registers report the status and change the behavior of the processor. Since the processor core only supports M-mode and D-mode, Nios V/g processor implements the CSRs supported by these two modes.

Table 176. Control and Status Registers List

Number	Privilege	Name	Description
Floating-Point CSRs			
0x001	MRW	`fflags`	Floating-Point Accrued Exceptions. Refer to Floating-Point CSR Register Fields table.
0x002	MRW	`frm`	Floating-Point Dynamic Rounding Mode. Refer to Floating-Point CSR Register Fields table.
0x003	MRW	`fcsr`	Floating-Point Control and Status Register (`frm` and `fflags`). Refer to Floating-Point CSR Register Fields table.
Machine Trap Setup			
0x300	MRW	`mstatus`	Machine status register. Refer to Machine Status Register Fields table.
0x301	MRW	`misa`	ISA and extensions. Refer to Machine ISA Register Fields table.
0x304	MRW	`mie`	Machine interrupt-enable register. Refer to Machine Interrupt-Enable Register Fields table.
0x305	MRW	`mtvec`	Machine trap-handler base address. Refer to .Machine Trap-Handler Base Address Register Fields table.
Machine Trap Handling			
0x341	MRW	`mepc`	Machine exception program counter. Refer to Machine Exception Program Counter Register Fields table.
0x342	MRW	`mcause`	Machine trap cause. Refer to Machine Trap Cause Register Fields table.
0x343	MRW	`mtval`	Machine bad address or instruction. Refer to Machine Trap Value Register Fields table.

continued...

Number	Privilege	Name	Description
0x344	MRW	mip	Machine interrupt pending. Refer to Machine Interrupt-Pending Register Fields table.
0x348	MRW	mtval2	Extended ECC information register. Refer to Machine Trap Value 2 Register Fields table.
Trigger Registers			
0x7A0	MRW	tselect	Trigger select. Refer to Trigger Select Register Fields table.
0x7A1	MRW	tdata1 (mcontrol)	Trigger data 1 (Match Control). Refer to Trigger Data 1 (Match Control) Register Fields table.
0x7A2	MRW	tdata2	Trigger data 2. Refer to Trigger Data 2 Register Fields table.
0x7A4	MRO	tinfo	Trigger info. Refer to Trigger Info Register Fields table.
Debug Mode Registers			
0x7B0	DRW	dcsr	Debug control and status register. Refer to Debug Control and Status Register Fields table.
0x7B1	DRW	dpc	Debug Program Counter. Refer to Debug Program Counter Register Fields table.
Shadow Mode Registers			
0x7C4	MRW	msrfstatus	Shadow register file status CSR. Refer to Shadow Register File Status CSR Fields table.
0x7C5	MRW	mrdpsrf	Read previous shadow register CSR. Refer to Read Previous Shadow Register CSR Fields table.
0x7C6	MRW	mwrpsrf	Write previous shadow register CSR. Refer to Write Previous Shadow Register CSR Fields table.
Machine Information Register			
0xF11	MRO	mvendorid	Vendor ID. Refer to Vendor ID Register Fields table.
0xF12	MRO	marchid	Architecture ID. Refer to Architecture ID Register Fields table.
0xF13	MRO	mimpid	Implementation ID. Refer to Implementation ID Register Fields table.
0xF14	MRO	mhartid	Hardware thread ID. Refer to Hardware Thread ID Register Fields table.

Table 177. Memory-mapped Registers

Memory Mapped Address Number	Privilege	Name	Description
ALT_CPU_MTIME_OFFSET (Refer to the system.h.)	MRW	mtime	Machine timer. Refer to Machine Timer Register Fields table.
ALT_CPU_MTIME_OFFSET + 0x8 (Refer to the system.h.)	MRW	mtimecmp	Machine timer compare register. Refer to Machine Timer Compare Register Fields table.
ALT_CPU_MTIME_OFFSET + 0x10 (Refer to the system.h.)	MRW	msip	Custom memory-mapped register to assert software interrupt into Nios V processor. Refer to Machine Software Interrupt Register Fields table.

Related Information

The RISC-V® Instruction Set Manual Volume II: Privileged Architecture
More information about M-mode CSR and their respective fields.

4.4.2.1. Control and Status Register Field (CLINT)

The value in the each CSR registers determines the state of the Nios V/g processor. The field descriptions are based on the RISC-V specification.

All CSRs can be accessed by using `csrr*` instruction in Machine mode except for the Debug mode registers (0x7B0 and 0x7B1) which can only be accessed through Debug mode.

Table 178. Floating-Point CSR Register Fields

The `fcsr` CSR is a 32-bit read/write register that holds the accrued exception flags.

RISC-V CSR instructions can access `fflags` and `frm` field individually by specifying the CSR address (0x001 and 0x002) respectively.

When accessing `frm` field individually using the CSR address 0x002, the rounding mode is transferred as bits [2:0] of the destination register.

RNE (000) is the only supporting rounding mode. Writing other rounding mode into `frm` generates an illegal instruction exception.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Rounding Mode (`frm`)			Accrued Exceptions (`fflags`)				
								000			NV	DZ	OF	UF	NX

Table 179. Machine Status Register Fields

The `mstatus` register is a 32-bit read-write register that keeps track of and controls the hart's current operating state.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SD	0								0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	FS[1:0]		MPP[1:0]		0		0	MPIE	0	0	0	MIE	0	0	0

The following bitfields are read-only 0:

- Bit 22 (TSR = 0): S-mode is not supported
- Bit 21 (TW = 0): There are no modes less privileged than M-mode.
- Bit 20 (TVM = 0): S-mode is not supported
- Bit 19 (MXR = 0): S-mode is not supported
- Bit 18 (SUM = 0): S-mode and U-mode are not supported
- Bit 17 (MPRV = 0): U-mode is not supported

- Bit 16 and Bit 15 (XS[1:0] = 0): S-mode is not supported
- Bit 10 and Bit 9 (VS[1:0] = 0): S-mode is not supported
- Bit 8 (SPP = 0): S-mode is not supported
- Bit 6 (UBE = 0): U-mode is not supported
- Bit 5 (SPIE = 0): S-mode is not supported
- Bit 1 (SIE = 0): S-mode is not supported

Table 180. Machine ISA Register Fields

The `misa` CSR is a read-write register reporting the ISA supported by the hart.

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

Table 181. Machine Interrupt-Enable Register Fields

The `mie` register is a 32-bit read-write register that contains interrupt enable bits. This core does not support supervisor or user modes. Thus, any interrupt related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MEIE	0	0	0	MTIE	0	0	0	MSIE	0	0	0

Table 182. Machine Trap-Handler Base Address Register Fields

The `mtvec` register is a 32-bit read/write register that holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE).

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

Table 183. Machine Exception Program Counter Register Fields

The `mepc` register is a 32-bit read-write register that holds the addresses of the instruction that was interrupted or that encountered the exception when a trap is taken into M-mode.

Bit Field													
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mepc														

Table 184. Machine Trap Cause Register Fields

The `mcause` register is a 32-bit read-write register that hold the code indicating the event that caused the trap when a trap is taken into M-mode.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Interrupt	Exception code [30:16]														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Exception code [15:0]															

Table 185. Machine Trap Value Register Fields

The `mtval` register is a 32-bit read-write register that is written with exception-specific information to assist software in handling the trap when a trap is taken into M-mode.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mtval														

Table 186. Machine Interrupt-Pending Register Fields

The `mip` register is a 32-bit read/write register containing information on pending interrupts. This core does not support supervisor or user modes. Thus, any interrupt related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Platform Interrupt[15:0]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MEIP	0	0	0	MTIP	0	0	0	MSIP	0	0	0

Table 187. Trigger Select Register Fields

The `tselect` register is a 32-bit read/write register that selects the current trigger is accessible by other trigger register.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
`tselect`														

Table 188. Machine Trap Value 2 Register Fields

The mtval2 register is a 32-bit read-write register that is written with exception-specific information to assist software in handling the trap when a trap is taken into M-mode.

This core does not support hypervisor mode. Thus, the Nios V/g processor implements this register for ECC exception only (No implementation on guest-page faults).

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mtval2														

Table 189. Trigger Data 1 (Match Control) Register Fields

The `tdata1 (mcontrol)` register is a 32-bit read/write register containing information on the trigger type, tdata registers accessibility, and trigger implementation. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
type=2				dmode	maskmax						hit	select	timing	sizelo	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
action				chain	match				m	0	0	0	execute	store	load

Table 190. Trigger Data 2 Register Fields

The `tdata2` register is a 32-bit read/write register containing the trigger-specific data.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
tdata2														

Table 191. Trigger Info Register Fields

The `tinfo` register is a 32-bit read-only register containing information on each possible tdata1.type.

Bit Field													
31	30	29	…	18	17	16	15	14	13	…	2	1	0
0							info						

Table 192. Debug Control and Status Register Fields

The `dcsr` CSR is a 32-bit read-write register containing information and status during D-mode. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
debugver				0											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ebreakm	0	0	0	stepie	stopcount	stoptime	cause			0	mprven	nmip	step	prv	

Table 193. Debug Program Counter Register Fields

Upon entry to D-mode, `dpc` CSR is updated with the virtual address of the next instruction to be executed.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
dpc														

Table 194. Shadow Register File Status CSR Fields

The `msrfstatus` is a 32-bits wide register containing shadow register status information..

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

Table 195. Read Previous Shadow Register CSR Fields

The `mrdpsrf` register facilitates copying a single register from previous shadow register sets to the active one.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mrdpsrf														

Table 196. Write Previous Shadow Register CSR Fields

The `mwrpsrf` register facilitates copying a single register from previous shadow register sets to the previous one.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
mwrpsrf														

Table 197. Vendor ID Register Fields

The `mvendorid` CSR is a 32-bit read-only register that provides the JEDEC manufacturer ID of the provider of the core.

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

Table 198. Architecture ID Register Fields

The `marchid` CSR is a 32-bit read-only register encoding the base microarchitecture of the hart.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Architecture ID														

Table 199. Implementation ID Register Fields

The `mimpid` CSR provides a unique encoding of the version of the processor implementation.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Implementation														

Table 200. Hardware Thread ID Register Fields

The `mhartid` CSR is a 32-bit read-only register that contains the integer ID of the hardware thread running the code

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Hart ID														

4.4.2.2. Memory Mapped Registers Fields

A RISC-V processor applies LOAD or STORE instruction to access the memory-mapped registers. In the context of Nios V processor HAL API, the LOAD or STORE instructions are referred as IORD or IOWR instruction respective.

The Nios V processor HAL API provides macro to read or write these registers in:

- `bsp\HAL\inc\intel_niosv.h` for machine timer-related registers. The field descriptions are based on the RISC-V specification.
- `bsp\HAL\inc\sys\msw_interrupt.h` for the custom Machine Software Interrupt register.

Table 201. Machine Timer Register Field

The `mtime` register is a 64-bit read-wite register that keeps track of the number of clock cycles.

Bit Field														
63	62	61	60	59	58	57	...	6	5	4	3	2	1	0
`mtime`														

Table 202. Machine Timer Compare Register Field

The `mtimecmp` register is a 64-bit read-wite register that holds a reference value. A machine timer interrupt becomes pending whenever mtime contains a value greater than or equal to mtimecmp,

Bit Field														
63	62	61	60	59	58	57	...	6	5	4	3	2	1	0
`mtimecmp`														

Table 203. Custom Machine Software Interrupt Register Field

The `msip` register is a 32-bit read-wite register that asserts the machine software interrupt. The machine software interrupt becomes pending when msip (bit field 0) is 1, and vice versa.

Bit Field														
31	30	29	28	27	26	25	...	6	5	4	3	2	1	0
Hardwired to 0														`msip`

4.4.3. Control and Status Registers (CSR) Mapping (CLIC)

Control and status registers report the status and change the behavior of the processor. Since the processor core only supports M-mode and D-mode, Nios V/g processor implements the CSRs supported by these two modes.

Table 204. Control and Status Registers List

Number	Privilege	Name	Description
Floating-Point CSRs			
0x001	MRW	`fflags`	Floating-Point Accrued Exceptions. Refer to Floating-Point CSR Register Fields table.
0x002	MRW	`frm`	Floating-Point Dynamic Rounding Mode. Refer to Floating-Point CSR Register Fields table.
0x003	MRW	`fcsr`	Floating-Point Control and Status Rtegister (`frm` and `fflags`). Refer to Floating-Point CSR Register Fieldsable.
Machine Trap Setup			
0x300	MRW	`mstatus`	Machine status register. Refer to Machine Status Register Fields table.
0x301	MRW	`misa`	ISA and extensions. Refer to Machine ISA Register Fields table.
0x304	MRW	`mie`	Machine interrupt-enable register (Inactive in CLIC mode). Refer to Machine Interrupt-enable Register table.
0x305	MRW	`mtvec`	Machine trap-handler base address. Refer to Machine Trap-Handler Base Address Register Fields (For CLIC mode)table.
0x307	MRW	`mtvt`	Machine trap-handler vector table base address. Refer to Machine Trap-Handler Vector Table Base Address Register Fields table.
Machine Trap Handling			
0x340	MRW	`mscratch`	Machine scratch register for trap handlers. Refer to Machine Scratch Register for Trap Handler Register Fields table.
0x341	MRW	`mepc`	Machine exception program counter. Refer to Machine Exception Program Counter Register Fields table.
0x342	MRW	`mcause`	Machine trap cause. Refer to Machine Trap Cause Register Fields (For CLIC mode) table.
0x343	MRW	`mtval`	Machine bad address or instruction. Refer to Machine Trap Value Register Fields table.
0x344	MRW	`mip`	Machine interrupt-pending register (Inactive in CLIC mode).
0x345	MRW	`mnxti`	Machine interrupt handler address and enable modifier. Refer to Machine Next Interrupt Handler Address and Interrupt Enable Register Field table.
0x347	MRW	`mintthresh`	Machine interrupt-level threshold. Refer to Machine Interrupt-Level Register Fields table.
0x349	MRW	`mscratchcswl`	Machine conditional scratch swap on level change. Refer to Machine Scratch Swap for Interrupt-Level Change Register Fields table.
0xFB1	MRW	`mintstatus`	Machine current interrupt levels. Refer to Machine Interrupt Status Register Fields table.
Indirect Access CSRs			
0x350	MRW	`miselect`	Machine Indirect Select. Refer to Machine Indirect Select Register Fields table.
0x351	MRW	`mireg`	Machine Indirect Alias. Refer to Machine Indirect Alias Register Fields table.

continued...

Number	Privilege	Name	Description
0x352	MRW	mireg2	Machine Indirect Alias 2. Refer to Machine Indirect Alias 2 Register Fields table.
Trigger Registers			
0x7A0	MRW	tselect	Trigger select. Refer to Trigger Select Register Fields table.
0x7A1	MRW	tdata1 (mcontrol)	Trigger data 1 (Match Control). Refer to Trigger Data 1 (Match Control) Register Fields table.
0x7A2	MRW	tdata2	Trigger data 2. Refer to Trigger Data 2 Register Fields table.
0x7A4	MRO	tinfo	Trigger info. Refer to Trigger Info Register Fields table.
Debug Mode Registers			
0x7B0	DRW	dcsr	Debug control and status register. Refer to Debug Control and Status Register Fields table.
0x7B1	DRW	dpc	Debug Program Counter. Refer to Debug Program Counter Register Fields table.
Shadow Registers CSRs			
0x7C4	MRW	msrfstatus	Shadow register file status CSR. Refer to Shadow register file status CSR Fields table.
0x7C5	MRW	mrdpsrf	Read previous shadow register CSR. Refer to Read previous shadow register CSR Fields table.
0x7C6	MRW	mwrpsrf	Write previous shadow register CSR. Refer to Write previous shadow register CSR Fields table.
Machine Information Register			
0xF11	MRO	mvendorid	Vendor ID. Refer to Vendor ID Register Fields table.
0xF12	MRO	marchid	Architecture ID. Refer to Architecture ID Register Fields table.
0xF13	MRO	mimpid	Implementation ID. Refer to Implementation ID Register Fields table.
0xF14	MRO	mhartid	Hardware thread ID. Refer to Hardware Thread ID Register Fields table .

Table 205. Memory-mapped Register

Memory-mapped Address Number	Privilege	Name	Description
ALT_CPU_MTIME_OFFSET (Refer to the system.h.)	MRW	mtime	Machine timer. Refer to Machine Timer Register Fields table.
ALT_CPU_MTIME_OFFSET (Refer to the system.h.)	MRW	mtimecmp	Machine timer compare register. Refer to Machine Timer Compare Register Fields table.
ALT_CPU_MTIME_OFFSET + 0x10 (Refer to the system.h.)	MRW	msip	Custom memory-mapped register to assert software interrupt into Nios V processor. Refer to Machine Software Interrupt Register Fields table.

Table 206. CLIC Registers

Privilege	Name	Description
MRW	clicintip	CLIC interrupt pending. Refer to CLIC Interrupt Pending Slice Fields table.
MRW	clicintie	CLIC interrupt enable. Refer to CLIC Interrupt Enable Slice Fields table.
MRW	clicintattr	CLIC interrupt attribute. Refer to CLIC Interrupt Attribute Slice Fields table.
MRW	clicintctl	CLIC interrupt input control. Refer to CLIC Interrupt Input Control Slice Fields table.

4.4.3.1. Control and Status Register Fields (CLIC)

The value in each CSR registers determines the state of the Nios V/g processor. The field descriptions are based on the RISC-V specification. You can access all CSRs by using `csrr*` instruction in Machine mode except for the Debug mode registers (0x7B0 and 0x7B1) which can only be accessed through Debug mode.

Table 207. Floating-Point CSR Register Fields

The `fcsr` CSR is a 32-bit read/write register that holds the accrued exception flags.

RISC-V CSR instructions can access `fflags` and `frm` field individually by specifying the CSR address (0x001 and 0x002) respectively.

When accessing `frm` field individually using the CSR address 0x002, the rounding mode is transferred as bits [2:0] of the destination register.

RNE (000) is the only supporting rounding mode. Writing other rounding mode into `frm` generates an illegal instruction exception.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Rounding Mode (`frm`)			Accured Exceptions (`fflags`)				
								000			NV	DZ	OF	UF	NX

Table 208. Machine Status Register Fields

The `mstatus` register is a 32-bit read-write register that keeps track of and controls the hart's current operating state.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SD	0								0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	FS[1:0]		MPP[1:0]		0		0	MPIE	0	0	0	MIE	0	0	0

The following bitfields are read-only 0:

- Bit 22 (TSR = 0): S-mode is not supported
- Bit 21 (TW = 0): There are no modes less privileged than M-mode.
- Bit 20 (TVM = 0): S-mode is not supported

- Bit 19 (MXR = 0): S-mode is not supported
- Bit 18 (SUM = 0): S-mode and U-mode are not supported
- Bit 17 (MPRV = 0): U-mode is not supported
- Bit 16 and Bit 15 (XS[1:0] = 0): S-mode is not supported
- Bit 10 and Bit 9 (VS[1:0] = 0): S-mode is not supported
- Bit 8 (SPP = 0): S-mode is not supported
- Bit 6 (UBE = 0): U-mode is not supported
- Bit 5 (SPIE = 0): S-mode is not supported
- Bit 1 (SIE = 0): S-mode is not supported

Table 209. Machine ISA Register Fields

The `misa` CSR is a read-write register reporting the ISA supported by the hart.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MXL[1:0]		0				Extension[25:0]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Extension[25:0] .															

Table 210. Machine Interrupt-Enable Register Fields

The `mie` register is a 32-bit read-write register that contains interrupt enable bits. In CLIC mode, the `mie` register is inactive, thus hardwired to 0. It is replaced by the CLIC interrupt enable register (`clicintie`)

Bit Field		
31	...	0
0		

Table 211. Machine Trap-Handler Base Address Register Fields (For CLIC mode)

The `mtvec` register is a 32-bit read/write register that holds trap CLIC configuration, consisting of a trap CLIC base address (base) and a CLIC mode (mode = 11). `submode` bit field is reserved for future use.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
base[31:6]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
base[31:6]										submode = 0				mode = 11	

Table 212. Machine Trap-Handler Vector Table Base Address Register Fields

The `mtvt` register is a 32-bit read-write register that holds the Machine Trap Vector base address for CLIC vectored interrupts.

Bit Field												
31	30	29	...	8	7	6	5	4	3	2	1	0
base[31:6]							Reserved.					

Table 213. Machine Scratch Register for Trap Handler Register Fields

The `mscratch` register is a 32-bit read-write register that holds a pointer to a machine-mode hart-local context space and swapped with a user register upon entry to an M-mode trap handler.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mscratch														

Table 214. Machine Exception Program Counter Register Fields

The `mepc` register is a 32-bit read-write register that holds the addresses of the instruction that was interrupted or that encountered the exception when a trap is taken into M-mode.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mepc														

Table 215. Machine Trap Cause Register Fields (For CLIC mode)

The `mcause` register is a 32-bit read-write register that holds the code indicating the event that caused the trap when a trap is taken into M-mode.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
interrupt	minhv	mpp[1:0] = 00		mpie	Reserved			mpil[7:0]							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				exccode[11:0]											

Table 216. Machine Trap Value Register Fields

The `mtval` register is a 32-bit read-write register that is written with exception-specific information to assist software in handling the trap.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mtval														

Table 217. Machine Interrupt-Pending Register Fields

The `mip` register is a 32-bit read/write register containing information on pending interrupts. In CLIC mode, the `mip` register is inactive, thus hardwired to 0. It is replaced by separate interrupt pending (`clicintip`).

Bit Field		
31	…	0
0		

Table 218. Machine Next Interrupt Handler Address and Interrupt Enable Register Field

The `mnxti` register is a 32-bit read-write register that improve the performance of handling back-to-back software vectored interrupts.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mnxti														

Table 219. Machine Interrupt-Level Register Fields

The `mintthresh` register is a 32-bit read-write register that holds an 8-bit field for the threshold level.

Bit Field													
31	30	29	…	9	8	7	6	5	4	3	2	1	0
Reserved = 0						threshold							

Table 220. Machine Scratch Swap for Interrupt-Level Change Register Fields

The `mscratchswl` register is a 32-bit read-write register that support faster swapping of the stack pointer between interrupt and non-interrupt code.

Bit Field													
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mscratchswl														

Table 221. Machine Interrupt Status Register Fields

The `mintstatus` register is a 32-bit read-write register that holds the active interrupt level.

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

Table 222. Machine Indirect Select Register Fields

The `miselect` register is a 32-bit read-write register that select the CLIC registers based on specific index.

Bit Field													
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
miselect														

Table 223. Machine Indirect Alias Register Fields

The `mireg` register is a 32-bit read-write register that hold the current value of the CLIC register for register read and write operation. Use `miselect` register to select which CLIC register to display. Refer to Packing of Indirectly Accessed CLIC Registers on page 91 for more information.

Bit Field													
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mireg														

Table 224. Machine Indirect Alias 2 Register Fields

The `mireg2` register is a 32-bit read-write register that hold the current value of the CLIC register for register read and write operation. Use `miselect` register to select which CLIC register to display. Refer to Packing of Indirectly Accessed CLIC Registers on page 91 for more information.

Bit Field													
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mireg2														

Table 225. Trigger Select Register Fields

The `tselect` register is a 32-bit read/write register that selects the current trigger is accessible by other trigger register.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
tselect														

Table 226. Trigger Data 1 (Match Control) Register Fields

The `tdata1 (mcontrol)` register is a 32-bit read/write register containing information on the trigger type, tdata registers accessibility, and trigger implementation. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
type = 2				dmode	maskmax						hit	select	timing	sizelo	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
action				chain	match				m	0	0	0	execute	store	load

Table 227. Trigger Data 2 Register Fields

The `tdata2` register is a 32-bit read/write register containing the trigger-specific data.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
tdata2														

Table 228. Trigger Info Register Fields

The `tinfo` register is a 32-bit read-only register containing information on each possible tdata1.type.

Bit Field													
31	30	29	…	18	17	16	15	14	13	…	2	1	0
0							info						

Table 229. Debug Control and Status Register Fields

The `dcsr` CSR is a 32-bit read-write register containing information and status during D-mode. This core does not support supervisor or user modes. Thus, any bitfield related to these modes is read-only 0.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
debugver				0											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ebreakm	0	0	0	stepie	stopcount	stoptime	cause			0	mprven	nmip	step	Prv	

Table 230. Debug Program Counter Register Fields

Upon entry to D-mode, `dpc` CSR is updated with the virtual address of the next instruction to be executed.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
dpc														

Table 231. Shadow Register File Status CSR Fields

The `msrfstatus` is a 32-bits wide register containing shadow register status information. Refer to Shadow Register on page 65r for more information.

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

Table 232. Read Previous Shadow Register CSR Fields

The *mrdpsrf* register facilitates copying a single register from previous shadow register sets to the active one. Refer to Shadow Register for more information.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mrdpsrf														

Table 233. Write Previous Shadow Register CSR Fields

The *mwrpsrf* register facilitates copying a single register from active shadow register sets to the previous one. Refer to 4.3.2. Shadow Register for more information.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
mwrpsrf														

Table 234. Vendor ID Register Fields

The `mvendorid` CSR is a 32-bit read-only register that provides the JEDEC manufacturer ID of the provider of the core.

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

Table 235. Architecture ID Register Fields

The `marchid` CSR is a 32-bit read-only register encoding the base microarchitecture of the hart.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
Architecture ID														

Table 236. Implementation ID Register Fields

The `mimpid` CSR provides a unique encoding of the version of the processor implementation.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
Implementation														

Table 237. Hardware Thread ID Register Fields

The `mhartid` CSR is a 32-bit read-only register that contains the integer ID of the hardware thread running the code.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
Hart ID														

4.4.3.2. Memory Mapped Registers Fields

A RISC-V processor applies LOAD or STORE instruction to access the memory-mapped registers. In the context of Nios V processor HAL API, the LOAD or STORE instructions are referred as IORD or IOWR instruction respective.

The Nios V processor HAL API provides macro to read or write these registers in:

- `bsp\HAL\inc\intel_niosv.h` for machine timer-related registers. The field descriptions are based on the RISC-V specification.
- `bsp\HAL\inc\sys\msw_interrupt.h` for the custom Machine Software Interrupt register.

Table 238. Machine Timer Register Field

The `mtime` register is a 64-bit read-wite register that keeps track of the number of clock cycles.

Bit Field														
63	62	61	60	59	58	57	…	6	5	4	3	2	1	0
`mtime`														

Table 239. Machine Timer Compare Register Field

The `mtimecmp` register is a 64-bit read-wite register that holds a reference value. A machine timer interrupt becomes pending whenever mtime contains a value greater than or equal to mtimecmp,

Bit Field														
63	62	61	60	59	58	57	…	6	5	4	3	2	1	0
`mtimecmp`														

Table 240. Custom Machine Software Interrupt Register Field

The `msip` register is a 32-bit read-wite register that asserts the machine software interrupt. The machine software interrupt becomes pending when msip (bit field 0) is 1, and vice versa.

Bit Field														
31	30	29	28	27	26	25	…	6	5	4	3	2	1	0
Hardwired to 0														`msip`

4.4.3.3. CLIC Control and Status Registers Fields

The value in each CLIC CSRs and Indirect Access CSRs registers determines the state of the Nios V/g processor. These registers can be accessed through Indirect Access CSRs. The field descriptions are based on the RISC-V specification.

Note: `mie` and `mip` are hardwired to zero in CLIC mode and both registers are replaced by separate interrupt enable (`clicintie`) and pending (`clicintip`) respectively.

Table 241. CLIC Interrupt Pending Slice Fields

The `clicintip` slice is a 1-bit read-write register slice that holds the interrupt pending bit.

Bit Field
0
clicintip

Table 242. CLIC Interrupt Enable Slice Fields

The `clicintie` slice is a 1-bit read-write register slice that holds the interrupt enable bit.

Bit Field
0
clicintie

Table 243. CLIC Interrupt Attribute Slice Fields

The `clicintattr` slice is an 8-bit read-write register slice that specifies various attributes for each interrupt.

Bit Field							
7	6	5	4	3	2	1	0
mode = 11		Reserved			polarity	trigger	shv.

Table 244. CLIC Interrupt Input Control Slice Fields

The `clicintctl` slice is an 8-bit read-write register slice that specifies interrupt level and priority.

Bit Field							
7	6	5	4	3	2	1	0
clicintctl							

4.5. Core Implementation

4.5.1. Instruction Set Reference

The Nios V/g processor is based on the RV32IMZicsr_Zicbom specification, optionally "F" extension. RV32IMZicsr_Zicbom includes:

- RV32I base integer instruction ("I" extension)
- Integer multiplication and division instructions ("M" extension)
- Control and status register instructions ("Zicsr" extension)
- Base cache management operation ISA ("CMO" extension)
- Optionally, single-precision floating-point instruction ("F" extension)

There are 6 types of instruction formats. They are R-type, I-type, S-type, B-type, U-type, and J-type.

Besides the 'CMO' extension from RISC-V, Altera also provides customized cache management instructions. Refer to *Data Cache* for more information.

Table 245. Instruction Formats (R-type)

Bit Field (R-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
funct7							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd			opcode								

Table 246. Instruction Formats (I-type)

Bit Field (I-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:0]												rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			rd			opcode								

Table 247. Instruction Formats (S-type)

Bit Field (S-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[11:5]							rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:0]			opcode								

Table 248. Instruction Formats (B-type)

Bit Field (B-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[12]	imm[10:5]						rs2					rs1			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rs1	funct3			imm[4:1]			imm[11]		opcode						

Table 249. Instruction Formats (U-type)

Bit Field (U-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[31:16]															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Table 250. Instruction Formats (J-type)

Bit Field (J-type)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
imm[20]	imm[10:1]										imm[11]	imm[19:16]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[15:12]				rd					opcode						

Related Information

- The RISC-V® Instruction Set Manual Volume I: Unprivileged ISA
  More information about the instruction set of the RV32I Base Integer, the 'M' standard extension and the Zicsr extension.
- Data Cache on page 106

5. Nios V Processor Reference Manual Archives

For the latest and previous versions of this user guide, refer to Nios® V Processor Reference Manual. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

IP versions are the same as the Quartus Prime Design Suite software versions up to v19.1. From Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

6. Document Revision History for the Nios V Processor Reference Manual

Document Version	Quartus Prime Version	Changes
2025.05.25	25.1	• Updated the following topics in the section *Nios V/c*: — *Processor Performance Benchmarks*. — *Processor Architecture* — *General Purpose Register File* — *Control and Status Registers* — *Memory and I/O Organization* • Updated the following topics in the section *Nios V/m Processor*: — *General-Purpose Register File* — *Memory and I/O Organization* — *Instruction Manager Port* — *Data Manager Port* • Updated the section *Nios V/g Processor*: — Updated the following topics: • *Processor Architecture* • *Floating-point Unit* • *Data Manager Port* — Added the following new topics: • *Shadow Register* • *Trap Controller (CLIC)*
2025.01.27	24.3.1	• Updated the section *Nios V/c*: — Updated the figure *Nios V/c Processor Reset Network* to add *cpu_resetreq.ack* signal. — Added new topic *Instruction Cycles* • Updated the section *Nios V/m and Nios V/g processors*: — Updated the figure *Nios V/m Processor Reset Network* to add *cpu_resetreq.ack* signal. — Added new topic *Instruction Cycles* — Edited the topic *Trap Controller* to *Trap Controller (CLINT)* and added the following subtopics: • *Related Control and Status Register* • *Direct Mode* • *Vectored Mode* — Updated the topics *Control and Status Registers (CSR) Mapping* • Added a new table about Memory-mapped registers in the topic *Control and Status Registers (CSR) Field* • Added new subtopic *Memory Mapped Registers Fields*. • Revised the topic *RISC-V based Debug Module* for Nios V/m and Nios V/g processors.
2024.10.07	24.3	• Updated the fMAX and Logic Size in the table *Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Pro Edition* for each Nios V/c, Nios V/m, and Nios V/g processors. • Updated the table *Benchmark Parameters for Quartus Prime Software* for each Nios V/c, Nios V/m, and Nios V/g processors. • Added a new topic *Choosing A Suitable Interface* for Nios V/c Processor and Nios V/m processors.

continued...

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel 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.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Document Version	Quartus Prime Version	Changes
		• Updated the topic *Nios V/g: Processor Architecture*: — Updated the figure *Nios V/g Processor Core Block Diagram* to add Branch Prediction and Lockstep Module. — Added new topics: • *Branch Prediction* • *Lockstep Modules* • Edited the topic *Error Correction Code* and split int into a new topic *ECC Interface*. — Added new topics: • *ECC Error Injection* • *Affected CSR during ECC Event* • *Branch Prediction* — Added `mtval2` as a new control and status register to the table *Control and Status Registers List* and the topic *Control and Status Register Field*. • Edited the following topics to add more clarity: — *Nios V/m: Processor: Trap Controller* — *Nios V/m: Halt from Debug Module* — *Nios V/m: Hardware/Software Interface* — *Nios V/g: Control and Status Registers*
2024.07.26	24.2	• Updated the following topics for each Nios V/c, Nios V/m, and Nios V/g chapters: — Updated the fMAX and Logic Size in the Table: *Processor Performance Benchmarks* for Quartus Prime Pro Edition. — Updated the toolchain version in the Table: *Benchmark Parameters forQuartus Prime Software*. — *Instruction Manager Port*. — *Data Manager Port*. — *Error Correction Code*. — *Instruction Set Reference*.
2024.04.01	24.1	• Updated the following tables: — *Nios V/c Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software.* — *Benchmark Parameters for Quartus Prime Software* for the Nios V/c, Nios V/m, and Nios V/g processors. — *Instruction Interface Signals* for the Nios V/c, Nios V/m, and Nios V/g processors. — *Data Interface Signals* for the Nios V/c, Nios V/m, and Nios V/g processors. — *Nios V/m Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software*. — *Nios V/g Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Software*.
2023.12.11	23.4	• Updated all tables related to *Processor Performance Benchmarks*. • Edited *Nios V/c Processor Core Block Diagram*. • Revised instruction set from RV32IAZicsr to RV32IZicsr. • Updated *Accessing Tightly Coupled Memory*. • Updated the note in *Using Tightly Coupled Memory Effectively* to revise the support for Instruction TCM.

continued...

Document Version	Quartus Prime Version	Changes
2023.10.02	23.3	• Added Nios V/c processor in the table *Nios V Processor Variants*. • Added new section *Nios V/c Processor*. • Updated section *Nios V/m Processor*: — Added new sub-topics *Pipelined* and *Non-pipelined* in the topics *Processor Performance Benchmarks* and *Processor Pipeline*. — Updated table *Nios V/m Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Pro Edition Software*. — Added ECC module in figure *Nios V/m Processor Core Block Diagram*. — Updated tables *Instruction Interface Signals* and *Data Interface Signals*. — Added new topic *Error Correction Code (ECC)*. • Updated section *Nios V/g Processor*: — Updated table *Nios V/g Processor Performance Benchmarks in Altera FPGA Devices for Quartus Prime Pro Edition Software*. — Added ECC module, Instruction Tightly Coupled Memory, Data Tightly Coupled Memory, and Floating Point Unit in figure *Nios V/g Processor Core Block Diagram*. — Added the following topics: • *Floating-Point Unit* • *IEEE 754 Exception Conditions* • *Floating Point Operations* • *Tightly Coupled Memory* • *Instruction and Data Tightly-Coupled Memory* • *Accessing Tightly-Coupled Memory* • *Using Tightly-Coupled Memory Effectively* • *Error Correction Code (ECC)* — Updated tables *Instruction Interface Signals* and *Data Interface Signals*. — Added Floating-Point CSRs in table *Control and Status Registers List*. — Added *Floating-Point CSR Register Fields* in *Control and Status Register Field*.
2023.05.26	23.1	Added a link to *AN 980: Nios V Processor Quartus Prime Software Support.*
2023.04.14	23.1	• Updated *Nios V/m Processor Performance Benchmarks*. • Added a new section *Nios V/g Processor*. • Updated product family name to "Intel Agilex® 7".

Document Version	Quartus Prime Version	IP Version	Changes
2022.10.31	22.1std	1.0.0	• Updated references from *Quartus Prime Pro Edition* to Quartus Prime to indicate support for both Pro and Standard Edition. • Added Table: *Nios V/m Processor Performance Benchmarks in Intel FPGA Devices for Intel Quartus Prime Standard Edition* in *Processor Performance Benchmarks* section.
2022.09.26	22.3	22.3.0	• Updated the values in Table: *Nios V/m Processor Performance Benchmarks in Intel FPGA Devices*. • Replaced Table: *Reset and Debug Signals* with new signals, types, and descriptions. • Updated the step to set exception address in section *Exception Controller*.

continued...

Document Version	Quartus Prime Version	IP Version	Changes
2022.08.01	22.2	21.3.0	• Edited the performance metric value in Table: *Architecture Performance*. • Added new sections — *Trigger* — *Typical Use Cases* • Edited the following sections: — *Reset and Debug Signals* — *RISC-V based Debug Module* — *Debug Mode* — *Halt from Debug Module* — *Control and Status Registers (CSR) Mapping* — *Control and Status Register Field*
2022.06.30	22.1	21.2.0	Added new section *Reset and Debug Signals*.
2022.03.28	21.4	21.1.1	Updated *RISC-V based Debug Module* section with details for Nios V processor.
2021.12.13	21.4	21.1.1	Updated IP version and Quartus Prime version.
2021.11.15	21.3	21.1.0	Edited Table: *Architecture Performance* in Section: *Processor Performance Benchmarks*. • Change *CoreMark* to *CoreMark/MHz Ratio* and updated the value to *0.32148*.
2021.10.04	21.3	21.1.0	Initial release.