Nios® V Processor Reference Manual

Updated for Intel® Quartus® Prime Design Suite: **21.4**

IP Version: **21.1.1**

Online Version

Send Feedback

UG-20343

ID: 683632

Version: 2022.03.28

Contents

1. About this Document

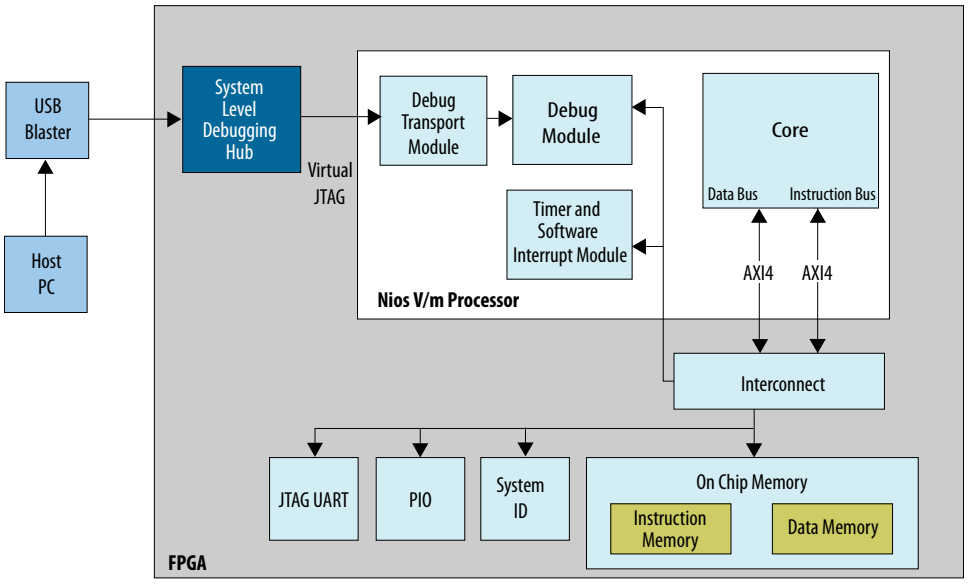
This document describes the Nios® V processor from the conceptual description to the details of implementation. The chapters in this document describe the hardware structure of the Nios V processor, including all the functional units, architecture, and the fundamentals of the Nios V processor hardware implementation.

1.1. Overview

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

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

Figure 1. Nios V/m Processor System using the Intel FPGA



Related Information

RISC-V Specification

2. Nios V/m Processor

The Nios V/m processor is a microcontroller core developed by Intel based on the RISC-V RV32IA instruction set and supports the functional units described in this document. This chapter describes the Nios V/m processor performance benchmark, the architecture, the programming model, and the instruction set. The Nios V/m processor is only available in the Intel® Quartus® Prime Pro Edition software version 21.3 release and later.

2.1. Processor Performance Benchmarks

Table 1. f_{max} (MHz)

Device Family	Nios V/m Processor
Intel Cyclone® 10	270.27
Intel Arria® 10	305.62
Intel Stratix® 10	361.93
Intel Agilex™	566.25

Table 2. Logic Size

Device Family	Nios V/m Processor
Intel Cyclone 10	1375
Intel Arria 10	1375
Intel Stratix 10	1580
Intel Agilex	1509

Table 3. Architecture Performance

Performance Metric	Nios V/m Processor
DMIPS/MHz Ratio	0.464
CoreMark/MHz Ratio	0.32148

Intel uses the following options for this benchmark:

- Maximum performance result based on 10 Seed Sweep from Intel Quartus Prime Design Suite software version 21.3.
- Fastest speed grade from each device family.
- Peripherals:
  - Nios V/m processor core
  - 4KB On-Chip Memory for Instruction Bus
  - 4KB On-Chip Memory for Data Bus
  - Avalon® Memory-Mapped Pipeline Bridge
- Intel Quartus Prime software compiler settings. Intel uses the same Intel Quartus Prime design example for FMAX and Logic Size benchmark but with different compiler setup:
  - FMAX: `superior_performance_optimized_placement_effort`
  - Logic Size: `area_aggressive`
- Toolchain:
  - Version Number:
    - xPack GNU RISC-V Embedded GCC, Linux 64-bit version: 10.1.1.0-1.1
    - CMake Version: 3.14.2
  - Compiler Configurations:
    - Compiler flags: `-03`
    - Assembler options: `-Wa -gdwarf2`
    - Compile options: `-Wall -Wformat-security -march=rv32ia -mabi=ilp32`

Disclaimer: Results may vary depending on the version of the Intel 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.

2.2. Processor Pipeline

The Nios V/m processor employs a 5-stages pipeline.

Table 4. Processor Pipeline Stages

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

continued...

Stage	Denotation	Function
Instruction Execute	E	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
Memory	M	• Memory and multicycle operations • Register file write • Branch redirection
Write Back	W	• Facilitates data dependency resolution by providing General Purpose Register value.

The Nios V/m processor implements the General Purpose Register file using M20K memory block. It takes one cycle to read from a M20K location. Hence, the F-stage initiates register file reads so General Purpose Register values are available in D-stage.

It takes two cycles to write to a M20K location. Hence, the M-stage initiates writes to a General Purpose Register and carries forward the value to W-stage, if there is a dependency to be resolved.

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

The pipeline can stall due to following reasons:

- 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.

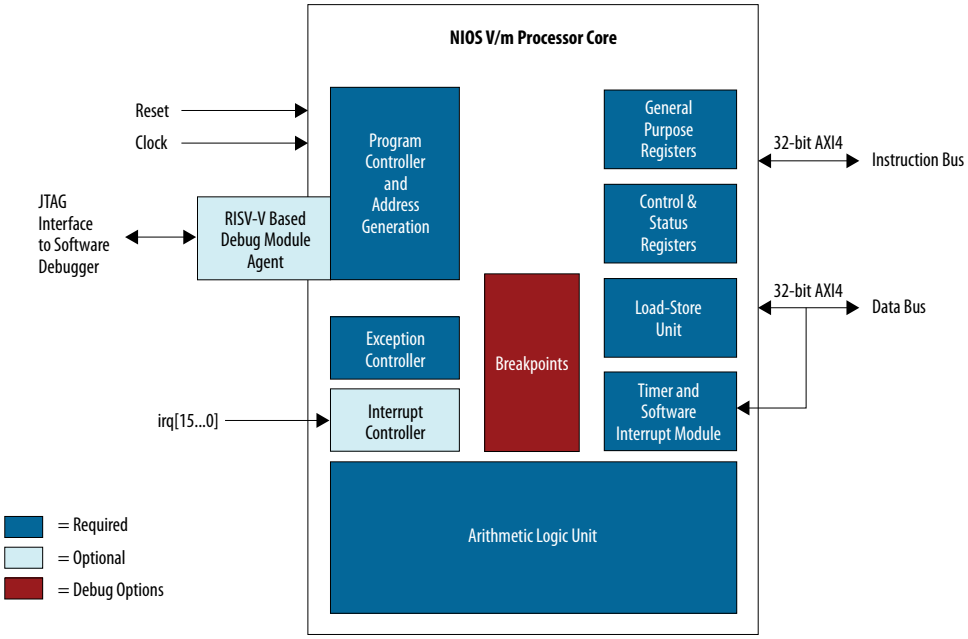
2.3. Processor Architecture

This section describes the hardware structure of the Nios V/m processor, including a discussion on all the functional units of the processor architecture and the fundamentals of the processor hardware implementation. 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)
- Exception Controller
- Interrupt Controller
- Instruction Bus
- Data Bus
- RISC-V based Debug Module

Figure 2. Nios V/m Processor Core Block Diagram



2.3.1. General Purpose Register File

Nios V/m processor implementation supports a flat register file, consisting of 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 duplicate register files so that two different operands for an instruction are available in a single cycle and write the same values to the memories.

2.3.2. 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. The ALU supports four fundamental data operations described in the following table.

Table 5. Supported Operations

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

For load/store instructions, Nios V/m processor uses ALU to calculate the memory address. For conditional control transfer instructions, Nios V/m processor uses the relational operations in ALU area to decide whether to take the branch or not.

2.3.3. 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. If a memory or multicycle instruction is pending in M-stage, CSR write does not take place until M-stage completes this instruction. The application's write to CSR is processed by the core after M-stage completes, only if M-stage completes without an exception. If an exception is generated during M-stage, all CSR and other pending instructions are flushed and exception handle will take over.

2.3.4. Exception Controller

Nios V/m processor architecture provides a simple exception controller to handle all exception types. Each exception, including internal hardware interrupts, causes the processor to transfer execution to an exception address. An exception handler at this address determines the cause of the exception and then executes an appropriate exception routine. You can set the exception address in the **Nios V/m Processor IP** parameterization. Nios V/m stores the address, which is writable, in Machine Trap Handler Base Address (`mtvec`) CSR register.

All exceptions are precise, which means that the processor completes all instructions preceding the faulting instruction and does not start the execution of instructions following the faulting instruction.

Table 6. Exceptions

Exception	Description
Instruction Address Misaligned	The core pipeline logic in F-stage detects the exception. This exception is flagged if the core fetched a Program Counter that is not aligned to a 32-bit word boundary.
Instruction Access Fault	Instruction read response signal detects this exception.
Illegal Instruction	Instruction decoder in D-stage flags this exception if an instruction word contains encoding for an unimplemented or undefined instruction. Control logic for CSR read/write flags this exception in E-stage if a CSR instruction accesses an unimplemented or undefined CSR.
Breakpoint	Instruction decoder flags the software breakpoint exception EBREAK in the D-stage.
Load Address Misaligned	The core for load/store unit in M-stage detects the misalignment. This exception is flagged if the data address is not aligned to the size of the data access.
Store Address Misaligned	
Load Access Fault	The core for data read/write response signal detects the exception.
Store Access Fault	
Env call from M-mode	Instruction decoder in D-stage detects the instruction.

2.3.5. Interrupt Controller

The Nios V/m processor implementation supports the following interrupts:

- Platform interrupts with 16 level-sensitive interrupt request (IRQ) inputs.
- Timer and Software interrupt - generated internally. You can access the timer interrupt register using the Timer and Software interrupt module interface by connecting to the data bus.

During an interrupt, the core writes the Program Counter of the attached instruction into the Machine Exception Program Counter (`mepc`) register. An interrupt is usually attached to the instruction in E-stage or in the preceding F or D pipeline stages. This is because the M-stage initiates the General Purpose Register. Thus, the core is not capable of retracting a memory instruction in M-stage. When an instruction in M-stage flags an exception while an interrupt is pending and ready to be serviced, the core will fetch and execute the exception instruction. If a memory or multicycle instruction is pending in M-stage, for example, the core is waiting for the response, the core does not flag an interrupt until the core receives a response for that instruction. Pending interrupts are flagged by their corresponding bits in Machine Interrupt-Pending (`mip`) register.

An interrupt is taken only when Machine Status Register (`mstatus`) bit 3 is asserted and bits corresponding to its pending interrupt in Machine Interrupt-pending (`mip`) register is asserted.

Table 7. Interrupt Control and Status Registers/Bits

Register	Description
`mstatus`	`mstatus`[3] / Machine Interrupt-enable (MIE) field • Global interrupt-enable bit for machine mode
`mie`	`mie`[7] / Machine Timer Interrupt-enable (MTIE) field • Timer interrupt-enable bit for machine mode `mie`[3] / Machine Software Interrupt-enable (MSIE) field • Software interrupt-enable bit for machine mode
`mip`	`mip`[7] / Machine Timer Interrupt-pending (MTIP) field • Timer interrupt-pending bit for machine mode `mip`[3] / Machine Software Interrupt-pending (MSIP) field • Software interrupt-pending bit for machine mode

2.3.5.1. Timer and Software Interrupt Module

The timer and software interrupt host the following registers:

- Machine Time (`mtime`) and Machine Time Compare (`mtimecmp`) registers for timer interrupt.
- Machine Software Interrupt-pending (`msip`) field for software interrupt.

The value of `mtime` increments after every clock cycle. When the value of `mtime` is greater or equal to the value of `mtimecmp`, the timer post the interrupt.

2.3.6. Memory and I/O Organization

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

Nios V/m processor systems are configurable. As a result, the memory and I/O organization varies from system to system. A Nios V/m processor core uses one or more of the following to provide memory and I/O access:

- 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.

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 halfwords 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.

2.3.6.1. Instruction and Data Buses

2.3.6.1.1. Instruction Manager Port

Nios V/m 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/m processor system is not applicable to the programs. Instruction address is always aligned to a 32-bit word boundary.

Table 8. Instruction Interface Signals

Interface	Signal	Role	Direction
Write Address Channel	`awaddr`	Unused	Output
	`awprot`	Unused	Output
	`awsize`	Unused	Output
	`awready`	Unused	Input
Write Data Channel	`wvalid`	Unused	Output
	`wdata`	Unused	Output
	`wstrb`	Unused	Output
	`wlast`	Unused	Output
	`wready`	Unused	Input
Write Response Channel	`bvalid`	Unused	Input

continued...

Interface	Signal	Role	Direction
	bres	Unused	Input
	bready	Unused	Output
Read Address Channel	araddr	Instruction Address (Program Counter)	Output
	arprot	Unused- tied off to constant value	Output
	arvalid	Instruction request valid	Output
	arsize	Constant 2- 4 bytes	Output
	arready	From subordinate/interconnect	Input
Read Data Channel	rdata	Instruction	Input
	rvalid	Instruction valid	Input
	rresp	Instruction response: Non-zero value denotes instruction access fault exception.	Input
	rready	Constant 1	Output

2.3.6.1.2. Data Manager Port

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

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

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 9. Data Interface Signals

Interface	Signal	Role	Direction
Write Address Channel	awaddr	Store address	Output
	awprot	Undefined- constant value	Output
	awvalid	Store valid	Output
	awsize	Store size- SB, SH, SW	Output
	awready	From memory/interconnect	Input
Write Data Channel	wvalid	Store valid	Output
	wdata	Store data	Output
	wstrb	Byte position in word	Output

continued...

Interface	Signal	Role	Direction
	wlast	Constant 1	Output
	wready	From memory/interconnect	Input
Write Response Channel	bvalid	Store done	Input
	bresp [1:0]	Store done status: Non-zero value denotes store access fault exception.	Input
	bready	Constant 1	Output
Read Address Channel	araddr	Load Address	Output
	arprot	Undefined- constant value	Output
	arvalid	Load Valid	Output
	arsize	Load size: LB, LH, LW	Output
	arready	From subordinate/interconnect	Input
Read Data Channel	rdata	Read data	Input
	rvalid	Read data valid	Input
	rresp	Load response status: Non-zero value denotes load access fault exception.	Input
	rready	Constant 1	Output

2.3.6.1.3. Address Map

The address map for memories and peripherals in a Nios V/m processor system is design dependent. There are four addresses that 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 which is stored in the `mvtec` register. `mvtime` and `mtimecmp` register controls the timer interrupt and the `msip` register bit controls the software interrupt.

2.3.7. 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
- Download programs to memory
- Start and stop execution
- Set software breakpoints and watchpoints
- Analyze registers and memory

2.3.7.1. Debug Mode

You can enter the debug mode, as specified in the RISC-V architecture specification, in two ways:

1. Halt from Debug Module
2. Software breakpoints

Debug Module selects Hardware Thread (Hart); which can be in one of the following four states:

1. Non-existent: Debug Module probes a hart which does not exist.
2. Unavailable: Reset or temporary shutdown.
3. Running: Normal operation outside of debug.
4. Halted: Hart is said to be halted when it is in debug mode.

2.3.7.2. Halt from Debug Module

Debugger can write to the `haltreq` bit in Debug Module Control (`dmcontrol`) register, which places the core in debug mode after some handshake.

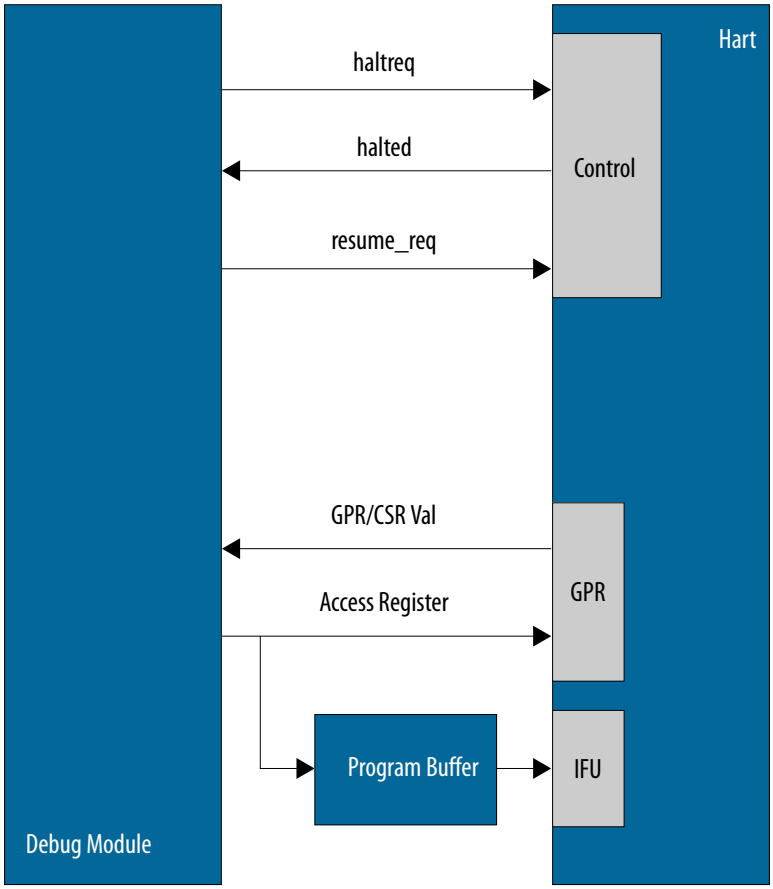
2.3.7.3. Debug Implementation

Assertion of `haltreq` bit send an asynchronous interrupt to the core logic. Core logic 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 in M-stage, the Program Counter writes to the Debug Program Counter.
- If the instruction in M-stage is not valid, then instruction in E-stage can be interrupted and so on and so forth.
- When there is no valid instruction in the pipe, the Program Counter for the next instruction to be executed writes to the Debug Program Counter.

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

Figure 3. Debug Module Block Diagram



2.3.7.4. Abstract Commands in Debug Mode

Nios V/m processor implements Access Register abstract command. The Access Register command allows read-write access to the processor registers including GPRs, CSRs, FP registers and Program Counter. The Access Register also allows program execution from program buffer. The debugger executes Access Register commands by writing into Abstract Command (`command`) register using the Access Register command encoding.

Table 10. Access Register Command Encoding

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
cmdtype								0	aarsize			aarpostincrement	postexec	transfer	write
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
regno															

Table 11. Fields Descriptions

Field	Role
cmdtype	Determine command type 0 : Indicates Access Register command.
aarsize	Specifies size of register access 2: Access the lowest 32 bit of register 3: Access the lowest 64 bit of register 4: Access the lowest 128 bit of register
aarpostincrement	0: No effect 1: regno is incremented 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 1: Copy data to register
regno	Register address to be accessed.

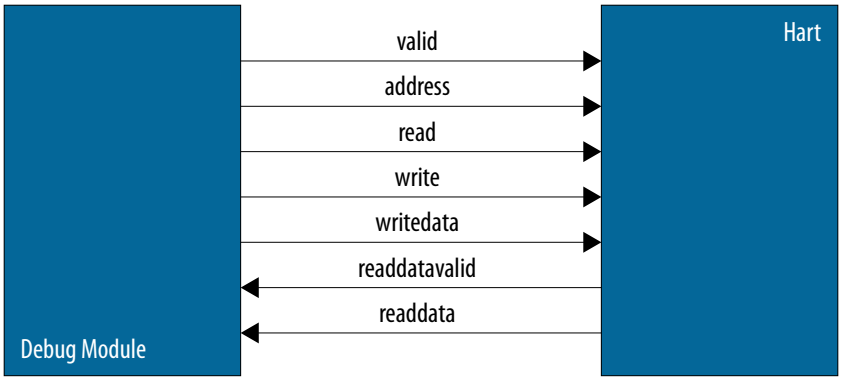
Note: The Nios V/m processor does not support abstract commands when hardware thread is not halted.

Table 12. Software Response to Unimplemented Commands

Field	State	
cmderr[10:8]	0	No error
	1	Busy
	2	Command not supported
	3	Exception - from program buffer instruction
	4	Command not executed because hart unavailable, or not in correct state to execute command.
	5	Abstract command failed due to bus error
	6	RSVD
	7	Command failed for other reasons.

Debug Module has the Abstract Control and Status CSR which includes the cmderr field. The cmderr field represents the current state of abstract command being executed. If the cmderr field is non-zero, writes to the command register are ignored. To clear the cmderr field, write 1 for every bit in the field.

Figure 4. Debug Module Interface Signals



Avalon memory-mapped interface implement the Register Access using request/response bus with Debug Module being the initiator and core being the responder. Address bus carries the register ID.

2.3.7.5. Hardware/Software Interface

Debug Module read or write to Debug Module registers, which initiate the interaction with the debugger. Each register has a fixed address as specified in the RISC-V Debug Support specification. Debugger can determine the register implementation status by write or read from the Debug Module registers. Unimplemented registers return 0 when read.

Debugger can check the status of the system by reading Debug Module Status (`dmstatus`) register. Debug Module Status register is read-only and provides status of the Debug Module and the selected harts.

You can access a specific hart by writing to the `hartsel` field in `dmcontrol`. Other fields in `dmcontrol` specify the action a debugger can take.

Halt Summary 0 (`haltsum0`) register reflects the status of a hart (halted/not halted). The LSB of this register can reflect whether hart is halted or not. Other bits is always 0. This is a read-only register of the debugger.

Related Information

RISC-V Debug Release
More information about the conceptual view of the states, refer to Section : Overview of States, Figure: Run/Halt Debug State Machine for single hart systems.

2.4. Programming Model

2.4.1. Privilege Levels

The privilege levels in Nios V/m 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

The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.11

2.4.1.1. Machine Mode (M-mode)

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

2.4.1.2. 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.

2.4.2. 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 13. Control and Status Registers List

Number	Privilege	Name	Description
Machine Information Register			
0xF11	MRO	`mvendorid`	Vendor ID. Refer to Table 14 on page 18.
0xF12	MRO	`marchid`	Architecture ID. Refer to Table 15 on page 18.
0xF13	MRO	`mimpid`	Implementation ID. Refer to Table 16 on page 18.
0xF14	MRO	`mhartid`	Hardware thread ID. Refer to Table 17 on page 18.
Machine Trap Setup			
0x300	MRW	`mstatus`	Machine status register. Refer to Table 18 on page 19.
0x301	MRW	`misa`	ISA and extensions. Refer to Table 19 on page 19.
0x304	MRW	`mie`	Machine interrupt-enable register. Refer to Table 20 on page 19.
0x305	MRW	`mtvec`	Machine trap-handler base address. Refer to Table 21 on page 19.
Machine Trap Handling			
0x341	MRW	`mepc`	Machine exception program counter. Refer to Table 22 on page 19.
0x342	MRW	`mcause`	Machine trap cause. Refer to Table 23 on page 20.
0x343	MRW	`mtval`	Machine bad address or instruction. Refer to Table 24 on page 20.
0x344	MRW	`mip`	Machine interrupt pending. Refer to Table 25 on page 20.

continued...

Number	Privilege	Name	Description
Debug Mode Registers			
0x7B0	DRW	dcsr	Debug control and status register. Refer to Table 26 on page 20.
0x7B1	DRW	dpc	Debug Program Counter. Refer to Table 27 on page 20.

Related Information

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

2.4.2.1. 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.

Table 14. 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 15. 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 16. 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 17. 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														

Table 18. 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	WPRI								TSR	TW	TVM	MXR	SUM	MPRV	XS[1:0]
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]		MPP[1:0]		WPRI		SPP	MPIE	WPRI	SPIE	UPIE	MIE	WPRI	SIE	UIE

Table 19. 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]		WLRL				Extension[25:0]									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Extension[25:0]															

Table 20. Machine Interrupt-Enable Register Fields

The `mie` register is a 32-bit read-write register that contains interrupt enable bits.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WPRI															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI				MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE

Table 21. 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 22. 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 23. 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 24. 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 25. Machine Interrupt-Pending Register Fields

The `mip` register is a 32-bit read/write register containing information on pending interrupts.

Bit Field															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WPRI															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI				MEIP	WPRI	SEIP	UEIP	MTIP	WPRI	STIP	UTIP	MSIP	WPRI	SSIP	USIP

Table 26. Debug Control and Status Register Fields

The `dcsr` CSR is a 32-bit read-write register containing information and status during D-mode

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	ebreaks	ebreaku	stepie	stopcount	stoptime	cause			0	mprven	nmip	step	prv	

Table 27. 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														

2.5. Core Implementation

2.5.1. Instruction Set Reference

The Nios V/m processor is based on the RV32IA specification, and there are 6 types of instruction formats. They are R-type, I-type, S-type, B-type, U-type, and J-type.

Table 28. 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 29. 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 30. 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 31. 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 32. 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 33. 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 and the "A" standard extension.

3. Document Revision History for the Nios V Processor Reference Manual

Document Version	Intel Quartus Prime Version	IP Version	Changes
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 Intel 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.