Nios II Core Implementation Details

This document describes all of the Nios[®] II processor core implementations available at the time of publishing. This document describes only implementation-specific features of each processor core. All cores support the Nios II instruction set architecture.

For more information regarding the Nios II instruction set architecture, refer to the *Instruction Set Reference* chapter of the *Nios II Processor Reference Handbook*.

For common core information and details on a specific core, refer to the appropriate section:

Table 1: Nios II Processor Cores

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz[1]	0.15	0.74	1.16
	Max. DMIPS	31	127	218
	Max. f_{MAX}	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	Without MMU or MPU: < 1800 LEs; < 900 ALMs With MMU: < 3000 LEs; < 1500 ALMs With MPU: < 2400 LEs; < 1200 ALMs
Pipeline		1 stage	5 stages	6 stages
External Address Space		2 GB	2 GB	2 GB without MMU 4 GB with MMU
Instruction Bus	Cache	–	512 bytes to 64 KB	512 bytes to 64 KB
	Pipelined Memory Access	–	Yes	Yes
	Branch Prediction	–	Static	Dynamic
	Tightly-Coupled Memory	–	Optional	Optional

ISO 9001:2008 Registered

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Data Bus	Cache	–	–	512 bytes to 64 KB
	Pipelined Memory Access	–	–	–
	Cache Bypass Methods	–	–	• I/O instructions • Bit-31 cache bypass • Optional MMU
	Tightly-Coupled Memory	–	–	Optional
Arithmetic Logic Unit	Hardware Multiply	–	3-cycle$^{(2)}$	1-cycle$^{(2)}$
	Hardware Divide	–	Optional	Optional
	Shifter	1 cycle-per-bit	3-cycle shift$^{(2)}$	1-cycle barrel shifter (2)
JTAG Debug Module	JTAG interface, run control, software breakpoints	Optional	Optional	Optional
	Hardware Breakpoints	–	Optional	Optional
	Off-Chip Trace Buffer	–	Optional	Optional
Memory Management Unit		–	–	Optional
Memory Protection Unit		–	–	Optional
Exception Handling	Exception Types	Software trap, unimplemented instruction, illegal instruction, hardware interrupt	Software trap, unimplemented instruction, illegal instruction, hardware interrupt	Software trap, unimplemented instruction, illegal instruction, supervisor-only instruction, supervisor-only instruction address, supervisor-only data address, misaligned destination address, misaligned data address, division error, fast TLB miss, double TLB miss, TLB permission violation, MPU region violation, internal hardware interrupt, external hardware interrupt, nonmaskable interrupt
	Integrated Interrupt Controller	Yes	Yes	Yes
	External Interrupt Controller Interface	No	No	Optional
Shadow Register Sets		No	No	Optional, up to 63
User Mode Support		No; Permanently in supervisor mode	No; Permanently in supervisor mode	Yes; When MMU or MPU present
Custom Instruction Support		Yes	Yes	Yes
ECC support		No	No	Yes

Related Information

[Instruction Set Reference](#)

---

[1] DMIPS performance for the Nios II/s and Nios II/f cores depends on the hardware multiply option.

Device Family Support

All Nios II cores provide the same support for target Altera® device families.

Table 2: Device Family Support

Device Family	Support
Arria® GX	Final
Arria II GX	Final
Arria II GZ	Final
Arria V	Final
Arria V GZ	Final
Cyclone III	Final
Cyclone III LS	Final
Cyclone IV GX	Final
Cyclone IV E	Final
Cyclone V	Final
Stratix III	Final
Stratix IV E	Final
Stratix IV GT	Final
Stratix IV GX	Final
Stratix V	Final
Other device families	No support

Preliminary support—The core is verified with preliminary timing models for this device family. The core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.

Final support—The core is verified with final timing models for this device family. The core meets all functional and timing requirements for the device family and can be used in production designs.

Nios II/f Core

The Nios II/f fast core is designed for high execution performance. Performance is gained at the expense of core size. The base Nios II/f core, without the memory management unit (MMU) or memory protection unit (MPU), is approximately 25% larger than the Nios II/s core. Altera designed the Nios II/f core with the following design goals in mind:

---

[2] Multiply and shift performance depends on the hardware multiply option you use. If no hardware multiply option is used, multiply operations are emulated in software, and shift operations require one cycle per bit. For details, refer to the arithmetic logic unit description for each core.

- Maximize the instructions-per-cycle execution efficiency
- Optimize interrupt latency
- Maximize f_{MAX} performance of the processor core

The resulting core is optimal for performance-critical applications, as well as for applications with large amounts of code and/or data, such as systems running a full-featured operating system.

Overview

The Nios II/f core:

- Has separate optional instruction and data caches
- Provides optional MMU to support operating systems that require an MMU
- Provides optional MPU to support operating systems and runtime environments that desire memory protection but do not need virtual memory management
- Can access up to 2 GB of external address space when no MMU is present and 4 GB when the MMU is present
- Supports optional external interrupt controller (EIC) interface to provide customizable interrupt prioritization
- Supports optional shadow register sets to improve interrupt latency
- Supports optional tightly-coupled memory for instructions and data
- Employs a 6-stage pipeline to achieve maximum DMIPS/MHz
- Performs dynamic branch prediction
- Provides optional hardware multiply, divide, and shift options to improve arithmetic performance
- Supports the addition of custom instructions
- Optional ECC support for internal RAM blocks (instruction cache, MMU TLB, and register file)
- Supports the JTAG debug module
- Supports optional JTAG debug module enhancements, including hardware breakpoints and real-time trace

The following sections discuss the noteworthy details of the Nios II/f core implementation. This document does not discuss low-level design issues or implementation details that do not affect Nios II hardware or software designers.

Arithmetic Logic Unit

The Nios II/f core provides several arithmetic logic unit (ALU) options to improve the performance of multiply, divide, and shift operations.

Multiply and Divide Performance

The Nios II/f core provides the following hardware multiplier options:

- DSP Block—Includes DSP block multipliers available on the target device. This option is available only on Altera FPGAs that have DSP Blocks.
- Embedded Multipliers—Includes dedicated embedded multipliers available on the target device. This option is available only on Altera FPGAs that have embedded multipliers.
- Logic Elements—Includes hardware multipliers built from logic element (LE) resources.
- None—Does not include multiply hardware. In this case, multiply operations are emulated in software.

The Nios II/f core also provides a hardware divide option that includes LE-based divide circuitry in the ALU.

Including an ALU option improves the performance of one or more arithmetic instructions.

Note: The performance of the embedded multipliers differ, depending on the target FPGA family.

Table 3: Hardware Multiply and Divide Details for the Nios II/f Core

ALU Option	Hardware Details	Cycles per Instruction	Result Latency Cycles	Supported Instructions
No hardware multiply or divide	Multiply and divide instructions generate an exception	–	–	None
Logic elements	ALU includes 32 x 4-bit multiplier	11	+2	`mul, muli`
DSP block on Stratix III families	ALU includes 32 x 32-bit multiplier	1	+2	`mul, muli, mulxss, mulxsu, mulxuu`
Embedded multipliers on Cyclone III families	ALU includes 32 x 16-bit multiplier	5	+2	`mul, muli`
Hardware divide	ALU includes multicycle divide circuit	4 – 66	+2	`div, divu`

The cycles per instruction value determines the maximum rate at which the ALU can dispatch instructions and produce each result. The latency value determines when the result becomes available. If there is no data dependency between the results and operands for back-to-back instructions, then the latency does not affect throughput. However, if an instruction depends on the result of an earlier instruction, then the processor stalls through any result latency cycles until the result is ready.

In the following code example, a multiply operation (with 1 instruction cycle and 2 result latency cycles) is followed immediately by an add operation that uses the result of the multiply. On the Nios II/f core, the `addi` instruction, like most ALU instructions, executes in a single cycle. However, in this code example, execution of the `addi` instruction is delayed by two additional cycles until the multiply operation completes.

```
mul r1, r2, r3          ; r1 = r2 * r3
addi r1, r1, 100        ; r1 = r1 + 100 (Depends on result of mul)
```

In contrast, the following code does not stall the processor.

```
mul r1, r2, r3          ; r1 = r2 * r3
or r5, r5, r6           ; No dependency on previous results
or r7, r7, r8           ; No dependency on previous results
addi r1, r1, 100        ; r1 = r1 + 100 (Depends on result of mul)
```

Shift and Rotate Performance

The performance of shift operations depends on the hardware multiply option. When a hardware multiplier is present, the ALU achieves shift and rotate operations in three or four clock cycles. Otherwise, the ALU includes dedicated shift circuitry that achieves one-bit-per-cycle shift and rotate performance.

Refer to the "Instruction Execution Performance for Nios II/f Core" table in the "Instruction Performance" section for details.

Memory Access

The Nios II/f core provides optional instruction and data caches. The cache size for each is user-definable, between 512 bytes and 64 KB.

The memory address width in the Nios II/f core depends on whether the optional MMU is present. Without an MMU, the Nios II/f core supports the bit-31 cache bypass method for accessing I/O on the data master port. Therefore addresses are 31 bits wide, reserving bit 31 for the cache bypass function. With an MMU, cache bypass is a function of the memory partition and the contents of the translation lookaside buffer (TLB). Therefore bit-31 cache bypass is disabled, and 32 address bits are available to address memory.

Instruction and Data Master Ports

The instruction master port is a pipelined Avalon® Memory-Mapped (Avalon-MM) master port. If the core includes data cache with a line size greater than four bytes, then the data master port is a pipelined Avalon-MM master port. Otherwise, the data master port is not pipelined.

The instruction and data master ports on the Nios II/f core are optional. A master port can be excluded, as long as the core includes at least one tightly-coupled memory to take the place of the missing master port.

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

Support for pipelined Avalon-MM transfers minimizes the impact of synchronous memory with pipeline latency. The pipelined instruction and data master ports can issue successive read requests before prior requests complete.

Instruction and Data Caches

This section first describes the similar characteristics of the instruction and data cache memories, and then describes the differences.

Both the instruction and data cache addresses are divided into fields based on whether or not an MMU is present in your system.

Table 4: Cache Byte Address Fields

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

Table 5: Cache Virtual Byte Address Fields

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
													line		

Bit Fields															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
line											offset				

Table 6: Cache Physical Byte Address Fields

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
											offset				

Instruction Cache

The instruction cache memory has the following characteristics:

- Direct-mapped cache implementation.
- 32 bytes (8 words) per cache line.
- The instruction master port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Critical word first.
- Virtually-indexed, physically-tagged, when MMU present.

The size of the tag field depends on the size of the cache memory and the physical address size. The size of the line field depends only on the size of the cache memory. The offset field is always five bits (i.e., a 32-byte line). The maximum instruction byte address size is 31 bits in systems without an MMU present. In systems with an MMU, the maximum instruction byte address size is 32 bits and the tag field always includes all the bits of the physical frame number (PFN).

The instruction cache is optional. However, excluding instruction cache from the Nios II/f core requires that the core include at least one tightly-coupled instruction memory.

Data Cache

The data cache memory has the following characteristics:

- Direct-mapped cache implementation
- Configurable line size of 4, 16, or 32 bytes
- The data master 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)
- Virtually-indexed, physically-tagged, when MMU present

The size of the tag field depends on the size of the cache memory and the physical address size. The size of the line field depends only on the size of the cache memory. The size of the offset field depends on the line size. Line sizes of 4, 16, and 32 bytes have offset widths of 2, 4, and 5 bits respectively. The maximum data byte address size is 31 bits in systems without an MMU present. In systems with an MMU, the maximum data byte address size is 32 bits and the tag field always includes all the bits of the PFN.

The data cache is optional. If the data cache is excluded from the core, the data master port can also be excluded.

The Nios II instruction set provides several different instructions to clear the data cache. There are two important questions to answer when determining the instruction to use. Do you need to consider the tag field when looking for a cache match? Do you need to write dirty cache lines back to memory before clearing? Below the table lists the most appropriate instruction to use for each case.

Table 7: Data Cache Clearing Instructions

Instruction	Ignore Tag Field	Consider Tag Field
Write Dirty Lines	flushd	flushda
Do Not Write Dirty Lines	initd	initda

Note: The 4-byte line data cache implementation substitutes the flushd instruction for the flushda instruction and triggers an unimplemented instruction exception for the initda instruction. The 16-byte and 32-byte line data cache implementations fully support the flushda and initda instructions.

For more information regarding the Nios II instruction set, refer to the *Instruction Set Reference* chapter of the *Nios II Processor Reference Handbook*.

The Nios II/f core implements all the data cache bypass methods.

For information regarding the data cache bypass methods, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*

Mixing cached and uncached accesses to the same cache line can result in invalid data reads. For example, the following sequence of events causes cache incoherency.

1. The Nios II core writes data to cache, creating a dirty data cache line.
2. The Nios II core reads data from the same address, but bypasses the cache.

Note: Avoid mixing cached and uncached accesses to the same cache line, regardless whether you are reading from or writing to the cache line. If it is necessary to mix cached and uncached data accesses, flush the corresponding line of the data cache after completing the cached accesses and before performing the uncached accesses.

Related Information

- **Instruction Set Reference**
- **Processor Architecture**

Bursting

When the data cache is enabled, you can enable bursting on the data master port. Consult the documentation for memory devices connected to the data master port to determine whether bursting can improve performance.

Tightly-Coupled Memory

The Nios II/f core provides optional tightly-coupled memory interfaces for both instructions and data. A Nios II/f core can use up to four each of instruction and data tightly-coupled memories. When a tightly-coupled memory interface is enabled, the Nios II core includes an additional memory interface master port. Each tightly-coupled memory interface must connect directly to exactly one memory slave port.

When tightly-coupled memory is present, the Nios II core decodes addresses internally to determine if requested instructions or data reside in tightly-coupled memory. If the address resides in tightly-coupled memory, the Nios II core fetches the instruction or data through the tightly-coupled memory interface.

Software accesses tightly-coupled memory with the usual load and store instructions, such as `ldw` or `ldwio`.

Accessing tightly-coupled memory bypasses cache memory. The processor core functions as if cache were not present for the address span of the tightly-coupled memory. Instructions for managing cache, such as `initd` and `flushd`, do not affect the tightly-coupled memory, even if the instruction specifies an address in tightly-coupled memory.

When the MMU is present, tightly-coupled memories are always mapped into the kernel partition and can only be accessed in supervisor mode.

Memory Management Unit

The Nios II/f core provides options to improve the performance of the Nios II MMU.

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

Related Information
Programming Model

Micro Translation Lookaside Buffers

The translation lookaside buffer (TLB) consists of one main TLB stored in on-chip RAM and two separate micro TLBs (µTLB) for instructions µITLB) and data (µDTLB) stored in LE-based registers.

The TLBs have a configurable number of entries and are fully associative. The default configuration has 6 µDTLB entries and 4 µITLB entries. The hardware chooses the least-recently used µTLB entry when loading a new entry.

The µTLBs are not visible to software. They act as an inclusive cache of the main TLB. The processor firsts look for a hit in the µTLB. If it misses, it then looks for a hit in the main TLB. If the main TLB misses, the processor takes an exception. If the main TLB hits, the TLB entry is copied into the µTLB for future accesses.

The hardware automatically flushes the µTLB on each TLB write operation and on a `wrctl` to the `tlbmisc` register in case the process identifier (PID) has changed.

Memory Protection Unit

The Nios II/f core provides options to improve the performance of the Nios II MPU.

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

Related Information
Programming Model

Execution Pipeline

This section provides an overview of the pipeline behavior for the benefit of performance-critical applications. Designers can use this information to minimize unnecessary processor stalling. Most application programmers never need to analyze the performance of individual instructions.

The Nios II/f core employs a 6-stage pipeline.

Table 8: Implementation Pipeline Stages for Nios II/f Core

Stage Letter	Stage Name
F	Fetch
D	Decode
E	Execute
M	Memory
A	Align
W	Writeback

Up to one instruction is dispatched and/or retired per cycle. Instructions are dispatched and retired in order. Dynamic branch prediction is implemented using a 2-bit branch history table. The pipeline stalls for the following conditions:

- Multicycle instructions
- Avalon-MM instruction master port read accesses
- Avalon-MM data master port read/write accesses
- Data dependencies on long latency instructions (e.g., load, multiply, shift).

Pipeline Stalls

The pipeline is set up so that if a stage stalls, no new values enter that stage or any earlier stages. No "catching up" of pipeline stages is allowed, even if a pipeline stage is empty.

Only the A-stage and D-stage are allowed to create stalls.

The A-stage stall occurs if any of the following conditions occurs:

- An A-stage memory instruction is waiting for Avalon-MM data master requests to complete. Typically this happens when a load or store misses in the data cache, or a `flushd` instruction needs to write back a dirty line.
- An A-stage shift/rotate instruction is still performing its operation. This only occurs with the multicycle shift circuitry (i.e., when the hardware multiplier is not available).
- An A-stage divide instruction is still performing its operation. This only occurs when the optional divide circuitry is available.
- An A-stage multicycle custom instruction is asserting its stall signal. This only occurs if the design includes multicycle custom instructions.

The D-stage stall occurs if an instruction is trying to use the result of a late result instruction too early and no M-stage pipeline flush is active. The late result instructions are loads, shifts, rotates, `rdctl`, multiplies (if hardware multiply is supported), divides (if hardware divide is supported), and multicycle custom instructions (if present).

Branch Prediction

The Nios II/f core performs dynamic branch prediction to minimize the cycle penalty associated with taken branches.

Instruction Performance

All instructions take one or more cycles to execute. Some instructions have other penalties associated with their execution. Late result instructions have two cycles placed between them and an instruction that uses

their result. Instructions that flush the pipeline cause up to three instructions after them to be cancelled. This creates a three-cycle penalty and an execution time of four cycles. Instructions that require Avalon-MM transfers are stalled until any required Avalon-MM transfers (up to one write and one read) are completed.

Table 9: Instruction Execution Performance for Nios II/f Core 4byte/line data cache

Instruction	Cycles	Penalties
Normal ALU instructions (e.g., add, cmplt)	1	
Combinatorial custom instructions	1	
Multicycle custom instructions	> 1	Late result
Branch (correctly predicted, taken)	2	
Branch (correctly predicted, not taken)	1	
Branch (mispredicted)	4	Pipeline flush
`trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, `wrprs`; illegal and unimplemented instructions	4 or 5	Pipeline flush
`call`, `jmpi`, `rdprs`	2	
`jmp`, `ret`, `callr`	3	
`rdctl`	1	Late result
`load` (without Avalon-MM transfer)	1	Late result
`load` (with Avalon-MM transfer)	> 1	Late result
`store` (without Avalon-MM transfer)	1	
`store` (with Avalon-MM transfer)	> 1	
`flushd`, `flushda` (without Avalon-MM transfer)	2	
`flushd`, `flushda` (with Avalon-MM transfer)	> 2	
`initd`, `initda`	2	
`flushi`, `initi`	4	
Multiply		Late result
Divide		Late result
Shift/rotate (with hardware multiply using embedded multipliers)	1	Late result
Shift/rotate (with hardware multiply using LE-based multipliers)	2	Late result
Shift/rotate (without hardware multiply present)	1 to 32	Late result
All other instructions	1	

For Multiply and Divide, the number of cycles depends on the hardware multiply or divide option. Refer to "Arithmetic Logic Unit" and "Instruction and Data Caches" s for details.

In the default Nios II/f configuration, instructions `trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, `wrprs` require four clock cycles. If any of the following options are present, they require five clock cycles:

- MMU
- MPU
- Division exception
- Misaligned load/store address exception
- Extra exception information
- EIC port
- Shadow register sets

Related Information

- **Data Cache** on page 7
- **Instruction and Data Caches** on page 6
- **Arithmetic Logic Unit** on page 4

Exception Handling

The Nios II/f core supports the following exception types:

- Hardware interrupts
- Software trap
- Illegal instruction
- Unimplemented instruction
- Supervisor-only instruction (MMU or MPU only)
- Supervisor-only instruction address (MMU or MPU only)
- Supervisor-only data address (MMU or MPU only)
- Misaligned data address
- Misaligned destination address
- Division error
- Fast translation lookaside buffer (TLB) miss (MMU only)
- Double TLB miss (MMU only)
- TLB permission violation (MMU only)
- MPU region violation (MPU only)

External Interrupt Controller Interface

The EIC interface enables you to speed up interrupt handling in a complex system by adding a custom interrupt controller.

The EIC interface is an Avalon-ST sink with the following input signals:

- `eic_port_valid`
- `eic_port_data`

Signals are rising-edge triggered, and synchronized with the Nios II clock input.

The EIC interface presents the following signals to the Nios II processor through the `eic_port_data` signal:

- Requested handler address (RHA)—The 32-bit address of the interrupt handler associated with the requested interrupt.
- Requested register set (RRS)—The six-bit number of the register set associated with the requested interrupt.
- Requested interrupt level (RIL)—The six-bit interrupt level. If RIL is 0, no interrupt is requested.
- Requested nonmaskable interrupt (RNMI) flag—A one-bit flag indicating whether the interrupt is to be treated as nonmaskable.

Table 10: eic_port_data Signal

Bit Fields														
44	...													
RHA														
...	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RHA		RRS						RNMI	RIL					

Following Avalon-ST protocol requirements, the EIC interface samples `eic_port_data` only when `eic_port_valid` is asserted (high). When `eic_port_valid` is not asserted, the processor latches the previous values of RHA, RRS, RIL and RNMI. To present new values on `eic_port_data`, the EIC must transmit a new packet, asserting `eic_port_valid`. An EIC can transmit a new packet once per clock cycle.

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

Related Information
Embedded Peripherals IP User Guide

ECC

The Nios II/f core has the option to add ECC support for the following Nios II internal RAM blocks.

- Instruction cache

  - ECC errors (1, 2, or 3 bits) that occur in the instruction cache are recoverable; the Nios II processor flushes the cache line and reads from external memory instead of correcting the ECC error.
- Register file

  - 1 bit ECC errors are recoverable
  - 2 bit ECC errors are not recoverable and generate ECC exceptions
- MMU TLB

  - 1 bit ECC errors triggered by hardware reads are recoverable
  - 2 bit ECC errors triggered by hardware reads are not recoverable and generate ECC exception.
  - 1 or 2 bit ECC errors triggered by software reads to the TLBMISC register do not trigger an exception, instead, TLBMISC.EE is set to 1. Software must read this field and invalidate/overwrite the TLB entry.

The ECC interface is an Avalon-ST source with the output signal `ecc_event_bus`. This interface allows external logic to monitor ECC errors in the Nios II processor.

The `ecc_event_bus` contains the ECC error signals that are driven to 1 even if ECC checking is disabled in the Nios II processor (when `CONFIG.ECCEN` or `CONFIG.ECCEXC` is 0). The following table describes the ECC error signals.

Table 11: ECC Error Signals

Bit	Field	Description	Effect on Software	Available
0	EEH	ECC error exception while in exception handler mode (i.e., STATUS.EH = 1).	Likely fatal	Always
1	RF_RE	Recoverable (1 bit) ECC error in register file RAM	None	Always
2	RF_UE	Unrecoverable (2 bit) ECC error in register file RAM	Likely fatal	Always
3	ICTAG_RE	Recoverable (1, 2, or 3 bit) ECC error in instruction cache tag RAM	None	Instruction cache present
4	ICDAT_RE	Recoverable (1, 2, or 3 bit) ECC error in instruction cache data RAM.	None	Instruction cache present
5	Reserved			
6	Reserved			
7	Reserved			
8	Reserved			
9	Reserved			
10	Reserved			
11	Reserved			
12	Reserved			
13	Reserved			
14	Reserved			
15	Reserved			
16	Reserved			
17	Reserved			
18	Reserved			
19	TLB_RE	Recoverable (1 bit) ECC error in TLB RAM (hardware read of TLB)	None	MMU present
20	TLB_UE	Unrecoverable (2 bit) ECC error in TLB RAM (hardware read of TLB)	Possibly fatal	MMU present
21	TLB_SW	Software-triggered (1, 2, or 3 bit) ECC error in software read of TLB	Possibly fatal	MMU present
22	Reserved			
23	Reserved			

Bit	Field	Description	Effect on Software	Available
24	Reserved			
25	Reserved			
26	Reserved			
27	Reserved			
28	Reserved			
29	Reserved			

JTAG Debug Module

The Nios II/f core supports the JTAG debug module to provide a JTAG interface to software debugging tools. The Nios II/f core supports an optional enhanced interface that allows real-time trace data to be routed out of the processor and stored in an external debug probe.

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

Nios II/s Core

The Nios II/s standard core is designed for small core size. On-chip logic and memory resources are conserved at the expense of execution performance. The Nios II/s core uses approximately 20% less logic than the Nios II/f core, but execution performance also drops by roughly 40%. Altera designed the Nios II/s core with the following design goals in mind:

- Do not cripple performance for the sake of size.
- Remove hardware features that have the highest ratio of resource usage to performance impact.

The resulting core is optimal for cost-sensitive, medium-performance applications. This includes applications with large amounts of code and/or data, such as systems running an operating system in which performance is not the highest priority.

Overview

The Nios II/s core:

- Has an instruction cache, but no data cache
- Can access up to 2 GB of external address space
- Supports optional tightly-coupled memory for instructions
- Employs a 5-stage pipeline
- Performs static branch prediction
- Provides hardware multiply, divide, and shift options to improve arithmetic performance
- Supports the addition of custom instructions
- Supports the JTAG debug module
- Supports optional JTAG debug module enhancements, including hardware breakpoints and real-time trace

The following sections discuss the noteworthy details of the Nios II/s core implementation. This document does not discuss low-level design issues or implementation details that do not affect Nios II hardware or software designers.

Arithmetic Logic Unit

The Nios II/s core provides several ALU options to improve the performance of multiply, divide, and shift operations.

Multiply and Divide Performance

The Nios II/s core provides the following hardware multiplier options:

- DSP Block—Includes DSP block multipliers available on the target device. This option is available only on Altera FPGAs that have DSP Blocks.
- Embedded Multipliers—Includes dedicated embedded multipliers available on the target device. This option is available only on Altera FPGAs that have embedded multipliers.
- Logic Elements—Includes hardware multipliers built from logic element (LE) resources.
- None—Does not include multiply hardware. In this case, multiply operations are emulated in software.

The Nios II/s core also provides a hardware divide option that includes LE-based divide circuitry in the ALU.

Including an ALU option improves the performance of one or more arithmetic instructions.

Note: The performance of the embedded multipliers differ, depending on the target FPGA family.

Table 12: Hardware Multiply and Divide Details for the Nios II/s Core

ALU Option	Hardware Details	Cycles per instruction	Supported Instructions
No hardware multiply or divide	Multiply and divide instructions generate an exception	–	None
LE-based multiplier	ALU includes 32 x 4-bit multiplier	11	`mul, muli`
Embedded multiplier on Stratix III families	ALU includes 32 x 32-bit multiplier	3	`mul, muli, mulxss, mulxsu, mulxuu`
Embedded multiplier on Cyclone III families	ALU includes 32 x 16-bit multiplier	5	`mul, muli`
Hardware divide	ALU includes multicycle divide circuit	4 – 66	`div, divu`

Shift and Rotate Performance

The performance of shift operations depends on the hardware multiply option. When a hardware multiplier is present, the ALU achieves shift and rotate operations in three or four clock cycles. Otherwise, the ALU includes dedicated shift circuitry that achieves one-bit-per-cycle shift and rotate performance.

Refer to the "Instruction Execution Performance for Nios II/s Core" table in the "Instruction Performance" section for details.

Related Information
Instruction Performance

Memory Access

The Nios II/s core provides instruction cache, but no data cache. The instruction cache size is user-definable, between 512 bytes and 64 KB. The Nios II/s core can address up to 2 GB of external memory. The Nios II architecture reserves the most-significant bit of data addresses for the bit-31 cache bypass method. In the Nios II/s core, bit 31 is always zero.

For information regarding data cache bypass methods, refer to the *Processor Architecture* chapter of the *Nios II Processor Reference Handbook*.

Related Information
Processor Architecture

Instruction and Data Master Ports

The instruction master port is a pipelined Avalon Memory-Mapped (Avalon-MM) master port. If the core includes data cache with a line size greater than four bytes, then the data master port is a pipelined Avalon-MM master port. Otherwise, the data master port is not pipelined.

The instruction and data master ports on the Nios II/f core are optional. A master port can be excluded, as long as the core includes at least one tightly-coupled memory to take the place of the missing master port.

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

Support for pipelined Avalon-MM transfers minimizes the impact of synchronous memory with pipeline latency. The pipelined instruction and data master ports can issue successive read requests before prior requests complete.

Instruction Cache

The instruction cache for the Nios II/s core is nearly identical to the instruction cache in the Nios II/f core. The instruction cache memory has the following characteristics:

- Direct-mapped cache implementation
- The instruction master port reads an entire cache line at a time from memory, and issues one read per clock cycle.
- Critical word first

Table 13: Instruction Byte Address Fields

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

The size of the tag field depends on the size of the cache memory and the physical address size. The size of the line field depends only on the size of the cache memory. The offset field is always five bits (i.e., a 32-byte line). The maximum instruction byte address size is 31 bits.

The instruction cache is optional. However, excluding instruction cache from the Nios II/s core requires that the core include at least one tightly-coupled instruction memory.

Tightly-Coupled Memory

The Nios II/s core provides optional tightly-coupled memory interfaces for instructions. A Nios II/s core can use up to four tightly-coupled instruction memories. When a tightly-coupled memory interface is enabled, the Nios II core includes an additional memory interface master port. Each tightly-coupled memory interface must connect directly to exactly one memory slave port.

When tightly-coupled memory is present, the Nios II core decodes addresses internally to determine if requested instructions reside in tightly-coupled memory. If the address resides in tightly-coupled memory, the Nios II core fetches the instruction through the tightly-coupled memory interface. Software does not require awareness of whether code resides in tightly-coupled memory or not.

Accessing tightly-coupled memory bypasses cache memory. The processor core functions as if cache were not present for the address span of the tightly-coupled memory. Instructions for managing cache, such as `initi` and `flushi`, do not affect the tightly-coupled memory, even if the instruction specifies an address in tightly-coupled memory.

Execution Pipeline

This section provides an overview of the pipeline behavior for the benefit of performance-critical applications. Designers can use this information to minimize unnecessary processor stalling. Most application programmers never need to analyze the performance of individual instructions.

The Nios II/s core employs a 5-stage pipeline.

Table 14: Implementation Pipeline Stages for Nios II/s Core

Stage Letter	Stage Name
F	Fetch
D	Decode
E	Execute
M	Memory
W	Writeback

Up to one instruction is dispatched and/or retired per cycle. Instructions are dispatched and retired in-order. Static branch prediction is implemented using the branch offset direction; a negative offset (backward branch) is predicted as taken, and a positive offset (forward branch) is predicted as not taken. The pipeline stalls for the following conditions:

- Multicycle instructions (e.g., shift/rotate without hardware multiply)
- Avalon-MM instruction master port read accesses
- Avalon-MM data master port read/write accesses
- Data dependencies on long latency instructions (e.g., load, multiply, shift operations)

Pipeline Stalls

The pipeline is set up so that if a stage stalls, no new values enter that stage or any earlier stages. No "catching up" of pipeline stages is allowed, even if a pipeline stage is empty.

Only the M-stage is allowed to create stalls.

The M-stage stall occurs if any of the following conditions occurs:

- An M-stage load/store instruction is waiting for Avalon-MM data master transfer to complete.
- An M-stage shift/rotate instruction is still performing its operation when using the multicycle shift circuitry (i.e., when the hardware multiplier is not available).
- An M-stage shift/rotate/multiply instruction is still performing its operation when using the hardware multiplier (which takes three cycles).
- An M-stage multicycle custom instruction is asserting its stall signal. This only occurs if the design includes multicycle custom instructions.

Branch Prediction

The Nios II/s core performs static branch prediction to minimize the cycle penalty associated with taken branches.

Instruction Performance

All instructions take one or more cycles to execute. Some instructions have other penalties associated with their execution. Instructions that flush the pipeline cause up to three instructions after them to be cancelled. This creates a three-cycle penalty and an execution time of four cycles. Instructions that require an Avalon-MM transfer are stalled until the transfer completes.

Table 15: Instruction Execution Performance for Nios II/s Core

Instruction	Cycles	Penalties
Normal ALU instructions (e.g., `add`, `cmplt`)	1	
Combinatorial custom instructions	1	
Multicycle custom instructions	> 1	
Branch (correctly predicted taken)	2	
Branch (correctly predicted not taken)	1	
Branch (mispredicted)	4	Pipeline flush
`trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, unimplemented	4	Pipeline flush
`jmp`, `jmpi`, `ret`, `call`, `callr`	4	Pipeline flush
`rdctl`	1	
`load`, `store`	> 1	
`flushi`, `initi`	4	
Multiply		
Divide		
Shift/rotate (with hardware multiply using embedded multipliers)	3	
Shift/rotate (with hardware multiply using LE-based multipliers)	4	

Instruction	Cycles	Penalties
Shift/rotate (without hardware multiply present)	1 to 32	
All other instructions	1	

Exception Handling

The Nios II/s core supports the following exception types:

- Internal hardware interrupt
- Software trap
- Illegal instruction
- Unimplemented instruction

JTAG Debug Module

The Nios II/s core supports the JTAG debug module to provide a JTAG interface to software debugging tools. The Nios II/s core supports an optional enhanced interface that allows real-time trace data to be routed out of the processor and stored in an external debug probe.

Nios II/e Core

The Nios II/e economy core is designed to achieve the smallest possible core size. Altera designed the Nios II/e core with a singular design goal: reduce resource utilization any way possible, while still maintaining compatibility with the Nios II instruction set architecture. Hardware resources are conserved at the expense of execution performance. The Nios II/e core is roughly half the size of the Nios II/s core, but the execution performance is substantially lower.

The resulting core is optimal for cost-sensitive applications as well as applications that require simple control logic.

Overview

The Nios II/e core:

- Executes at most one instruction per six clock cycles
- Can access up to 2 GB of external address space
- Supports the addition of custom instructions
- Supports the JTAG debug module
- Does not provide hardware support for potential unimplemented instructions
- Has no instruction cache or data cache
- Does not perform branch prediction

The following sections discuss the noteworthy details of the Nios II/e core implementation. This document does not discuss low-level design issues, or implementation details that do not affect Nios II hardware or software designers.

Arithmetic Logic Unit

The Nios II/e core does not provide hardware support for any of the potential unimplemented instructions. All unimplemented instructions are emulated in software.

The Nios II/e core employs dedicated shift circuitry to perform shift and rotate operations. The dedicated shift circuitry achieves one-bit-per-cycle shift and rotate operations.

Memory Access

The Nios II/e core does not provide instruction cache or data cache. All memory and peripheral accesses generate an Avalon-MM transfer. The Nios II/e core can address up to 2 GB of external memory. The Nios II architecture reserves the most-significant bit of data addresses for the bit-31 cache bypass method. In the Nios II/e core, bit 31 is always zero.

For information regarding data cache bypass methods, refer to the Processor Architecture chapter of the *Nios II Processor Reference Handbook*.

The Nios II/e core does not provide instruction cache or data cache. All memory and peripheral accesses generate an Avalon-MM transfer.

For information regarding data cache bypass methods, refer to the Processor Architecture chapter of the *Nios II Processor Reference Handbook*.

Related Information
Processor Architecture

Instruction Execution Stages

This section provides an overview of the pipeline behavior as a means of estimating assembly execution time. Most application programmers never need to analyze the performance of individual instructions.

Instruction Performance

The Nios II/e core dispatches a single instruction at a time, and the processor waits for an instruction to complete before fetching and dispatching the next instruction. Because each instruction completes before the next instruction is dispatched, branch prediction is not necessary. This greatly simplifies the consideration of processor stalls. Maximum performance is one instruction per six clock cycles. To achieve six cycles, the Avalon-MM instruction master port must fetch an instruction in one clock cycle. A stall on the Avalon-MM instruction master port directly extends the execution time of the instruction.

Table 16: Instruction Execution Performance for Nios II/e Core

Instruction	Cycles
Normal ALU instructions (e.g., `add`, `cmplt`)	6
All branch, `jmp`, `jmpi`, `ret`, `call`, `callr`	6
`trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, `rdctl`, unimplemented	6
All load word	6 + Duration of Avalon-MM read transfer
All load halfword	9 + Duration of Avalon-MM read transfer
All load byte	10 + Duration of Avalon-MM read transfer
All store	6 + Duration of Avalon-MM write transfer
All shift, all rotate	7 to 38
All other instructions	6

Instruction	Cycles
Combinatorial custom instructions	6
Multicycle custom instructions	6

Exception Handling

The Nios II/e core supports the following exception types:

- Internal hardware interrupt
- Software trap
- Illegal instruction
- Unimplemented instruction

JTAG Debug Module

The Nios II/e core supports the JTAG debug module to provide a JTAG interface to software debugging tools. The JTAG debug module on the Nios II/e core does not support hardware breakpoints or trace.

Document Revision History

Table 17: Document Revision History

Date	Version	Changes
April 2015	2015.04.02	Obsolete devices removed (Stratix II, Cyclone II).
February 2014	13.1.0	- Added information on ECC support - Removed HardCopy support information - Removed references to SOPC Builder
May 2011	11.0.0	Maintenance release.
December 2010	10.1.0	Maintenance release.
July 2010	10.0.0	- Updated device support nomenclature - Corrected HardCopy support information
November 2009	9.1.0	- Added external interrupt controller interface information. - Added shadow register set information.
March 2009	9.0.0	Maintenance release.
November 2008	8.1.0	Maintenance release.
May 2008	8.0.0	Added text for MMU and MPU.
October 2007	7.2.0	Added `jmpi` instruction to tables.
May 2007	7.1.0	- Added table of contents to Introduction section. - Added Referenced Documents section.

Date	Version	Changes
March 2007	7.0.0	Add preliminary Cyclone III device family support
November 2006	6.1.0	Add preliminary Stratix III device family support
May 2006	6.0.0	Performance for `flushi` and `initi` instructions changes from 1 to 4 cycles for Nios II/s and Nios II/f cores.
October 2005	5.1.0	Maintenance release.
May 2005	5.0.0	Updates to Nios II/f and Nios II/s cores. Added tightly-coupled memory and new data cache options. Corrected cycle counts for shift/rotate operations.
December 2004	1.2	Updates to Multiply and Divide Performance section for Nios II/f and Nios II/s cores.
September 2004	1.1	Updates for Nios II 1.01 release.
May 2004	1.0	Initial release.