Official Release

AndesCore™ NX25(F) Data Sheet

Document Number DS131-31
Date Issued 2023-12-12

Contact Information

Should you have any problems with the information contained herein, you may contact Andes Technology Corporation through:

Email — support@andestech.com
Website — https://es.andestech.com/eservice/

Please include the following information in your inquiries:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem.

General suggestions for improvements are welcome.

Revision History

Rev.	Rev. Date	Revised Content
3.1	2023-12-12	First Release for CPU Revision 4.3.0. • Update Section 10.5 by removing the usage limitation after design enhancement. • Update the description about FENCE instruction penalty cycle number in Section 16. • Add the description about supporting the programmable trigger for PLIC in Section 18.
3.0	2023-10-20	Second Release for CPU Revision 4.1.1. • Update the description of `mcountinhibit` in Section 15.4. • Add the following limitation and restriction to Section 10.5. – A FENCE instruction is required immediately after the RVA instruction for the concern, which the RVA instruction that misses D-Cache may generate store request twice.
2.9	2023-04-28	First Release for CPU Revision 4.1.1. • Update Section 10.7 about maximum number of outstanding AXI store transactions. • Enhance the description about wakeup sequence from WFI mode in Section 13.1. • Revise the description about ECC error handling in Section 14.3. • Update description about `mxstatus.IME/PIME` in Section 15.3.12. • Correct the instruction type in Table 95. • Update interface and configuration options for NCEPLDM200 in Section 20. • Remove obsolete signal from Table 132.

Continued on next page...

Revision History

Rev.	Rev. Date	Revised Content
2.8	2022-06-30	First Release for CPU Revision 4.1.0. • Describe RISC-V bit-manipulation extension in Section 2. • Add test_rvb in Section 23.3.8. • Add control register `mrvarch_cfg` in Section 15. • Upgrade TRACE interface to RISC-V Processor Trace. See Section 2.9.4 and Section 3.4 for details. • Clarify test_atcbmc300 description in Section 23. • Describe the latency of RISC-V bit-manipulation extension instructions in Section 16. • Add config option for synchronizer level in Section 2.2 and Section 2.11.1. • Clarify local memory usage constraints in Section 7.4. • Add the chapter Simulation with ACE in Section 21 and move the description of test_ace to *Andes Custom Extension User Manual*. • Correct the throughput of divide and remainder instructions in Section 16.
2.7	2022-03-12	First Release for CPU Revision 2.5.0. • Update the description and set requirements about the proper usage of BTB/RAS feature in Section 2.6.2. • Update the valid signals of trace interface in Section 3.4. • Correct the format of Table 16 and Table 21. • Describe memory ordering detail when D-Cache is not configured or off in Section 6.2. • Add interrupt priority table in Section 11.2.3. • Add user performance counters in Section 15.5. • Add no event selector in Table 85. • Update the description of `mip.IMECCI` in Section 15.3.11. • Correct the reset value of `mecc_code.CODE` in Section 15.10.3. • Correct the description of detail cause for imprecise exceptions in Section 11 and Section 15. • Modify reset tree figure in Section 17.3 to meet current designs. • Correct the description of AE350 SMU register in Section 17.11. • Remove the related descriptions about AE250.

Continued on next page...

Revision History

Rev.	Rev. Date	Revised Content
2.6	2021-05-28	First Release for CPU Revision 2.4.0. • Update Figure 4. • Update the signals description in Table 5 and Table 6. • Add Table 38 for AXI-ACM interface signals. • Update Table 45 with valid address range. • Add Section 9.9 to describe the interaction of CCTL operations and interrupts. • Add description about the limitation of the number of outstanding requests in Section 10.7. • Update the description in Section 15.3.10, Section 15.3.13, Section 15.7.5, Section 15.13.2, and Section 18.5.10. • Add notes about the CCTL operations in Section 15.10.9 and Section 15.10.11. • Update the description in Section 17.7.4 about the limitation of accessing ATCBMC200 internal register in AE350-AHB framework. • Add description about the configuration parameters of debug subsystem in Section 20.4. • Add the 128-bit bus width configuration in Table 137. • Update the description in Section 20.5.20 about the example sequence of the debug module accessing system bus.
2.5	2020-09-11	First Release for CPU Revision 2.2.0. • Update waveform for burst read access in local memory slave port in Figure 12. • Update descriptions and figure about interrupt latency in Section 18.6 and Figure 23. • Update the valid privilege modes in Table 3.
2.4	2020-05-29	First Release for CPU Revision 2.1.1.
2.3	2020-04-24	First Release for CPU Revision 2.1.0. (Internal Release) • Add the option of 4KiB size for I-Cache and D-Cache in Section 1.1, Section 2 and Section 3. • Increase numbers of device and write-through regions from 8 to 16 in Section 2.12 and Section 2.13.

Continued on next page. . .

Revision History

Rev.	Rev. Date	Revised Content
2.2	2019-11-25	First Release for CPU Revision 2.0.1. • Update the platform block diagrams in Section 1. • Update descriptions of the Device and Write-Through regions in Section 2. • Describe input signal dc_clk in Table 5. • Update descriptions of PMA in Section 6. • Update descriptions of SMU in Section 17.11.
2.1	2019-10-31	First Release for CPU Revision 2.0.0. • Describe CCTL operations in Section 9
2.0	2019-07-08	First Release for CPU Revision 1.8.1. • Document enhancement for Section 23.3.8
1.9	2019-06-30	First Release for CPU Revision 1.8.0. • Describe the sample simulation clock/reset generation in Section 4.3 • Update AE350 platform description in Section 1.4 and Section 26.1. • Update description about NCEPLIC100 configuration in Section 18.4.9 • Update description about NCEPLDM200 configuration in Section 2.10, Section 20.3 and Section 17.10
1.8	2019-05-10	First Release for CPU Revision 1.7.0. • Revise FENCE instruction behavior in Section 9.6 • Support optional BIU two ports structure. See Section 10.3. • Support optional NCEPLDM200 and debug subsystem. See Section 20. • Support dedicated clock and reset to LM for initialization and power control. See Section 8.7 • Describe the halt-on-reset related registers of NCEPLDM200 in Section 20.5. • Support Cadence Genus and mark Cadence RC as to-be-obsolete in Section 24.

Continued on next page...

Revision History

Rev.	Rev. Date	Revised Content
1.7	2019-03-31	Release for CPU Revision 1.6.0. • Enhance document content: – Describe enhanced vectored PLIC mode in Section 11 ,Section 15.3.6, Section 15.9.3 – Update `mxstatus.IME/PIME/DME` in Section 15.3.12 and `mip.IMECCI/BWEI/PMOVI` in Section 15.3.11 – Clarify tinfo description in Section 15.7.5 – Describe enhanced trigger in Section 15.7.8, Section 15.7.9, Section 15.7.10, Section 15.7.11 – Revise FENCE instruction behavior in Section 9.6 – Add descriptions in Section 3.2 – Add description about NCEPLDM200 enhancement in Section 20 – Add multi-core support of NCEPLMT100 in Section 19
1.6	2019-02-01	Second Release for CPU Revision 1.5.0. • Enhance document content: – Specify the CPU subsystem usage in Section 2.1 – Describe the behavior of WFI mode in Section 13.1 – Remove unused supervisor mode fields in Section 15
1.5	2018-12-14	First Release for CPU Revision 1.4.0. • Describe new features: – RISC-V N standard extension registers in Section 15.9 and Section 3 – Performance monitor access events in Table 85 – Native M-mode trigger in Section 15.7.6 – Halt-on-reset request in Section 3

Continued on next page...

Revision History

Rev.	Rev. Date	Revised Content
1.4	2018-09-14	Second Release for CPU Revision 1.3.0. • Describe ILM/DLM/D-Cache byte write enable signal assignment in Section 3. • Describe device cacheability region in Section 17. • Enhance PLIC vector mode extension description in Section 18. • Describe interface signals and configuration options of NCEPLIC100 and NCEPLMT100 in Section 18 and Section 19. • Add support of access memory abstract command of NCEPLDM200 in Section 20. • Describe interface signals and System Bus Access of NCEPLDM200 in Section 20.
1.3	2018-08-17	First Release for CPU Revision 1.3.0. • Change product name to NX25(F). • Describe trace interface in Section 3.4. • Describe low access latency mode in Section 10.6. • Add misaligned access, CCTL and floating-point extension in Section 1.1, Section 5.5 and Section 15. • Add new trigger control registers in Section 15. • Describe ECC/Performance-counter local interrupts in Section 15.

Continued on next page...

Revision History

Rev.	Rev. Date	Revised Content
1.2	2018-07-09	First Release for CPU Revision 1.2.0. • Describe new features: – Misaligned load/store accesses in Section 5.4 – AndeStar V5 performance counters in Table 85 – Write-through regions in Section 6 – Low access-latency AHB bus configuration in Section 2.5 – (NCEPLIC100) Support of asynchronous interrupt sources in Section 18.4 – AXI-based pre-integrated platform in Section 17 and relevant sections • Enlarge I/D-Cache Tag SRAM organization for lock, page-coloring and dirty bits in Section 3.11 and Section 3.12. • Update ACE signals in Section 3.15. • Describe effects of FENCE/FENCE.I instructions on caches in Section 9.6. • Describe interrupt latency in Section 18.6. • Renamed `rtc_clk` for the `mtime` counter to `mtime_clk` in Section 19. • Add or enhance system register descriptions in Section 15.3.1, Section 15.3.13, Section 15.10.7, Section 15.13.1, and Section 15.13.2. • Enhanced ACE description in Section 21 and Section 23.3.8.

Continued on next page. . .

Revision History

Rev.	Rev. Date	Revised Content
1.1	2018-01-31	First Release for CPU Revision 1.1.0. • Describe new features related to ACE support. – Add Andes Custom Extension (ACE) support in Section 1, Section 2 and Section 4.1. – Add ACE signals in Section 3. – Add ACE exceptions in Section 11. – Add ACE instructions latency in Section 16. – Add ACE related system register in Section 15.6.3, Section 15.8.5, Section 15.8.6 and Section 15.10.7. • Clarify the reset_vector value in Table 5. • Clarify the reset value of `mnvec` in Section 15.10.4. • Clarify the meaning of `mtvec.BASE` in Section 15.3.6. • Describe the behavior of the external interrupt pending signal (`tx_eip`) in Section 18. • Describe the behavior of interrupt claim and complete in Section 18.5.10. • Rename `mie.SBE` to `mie.BWE` and `mip.SBE` to `mip.BWE` in Section 15.3.5 and Section 15.3.11. • Describe `mmsc_cfg.EV5MPE` and `mmsc_cfg.LMSLVP` in Section 15.6.3. • Add quick start and NDSROM.dat sections in Section 23.4.
1.0	2017-12-12	First Release for CPU Revision 1.0.0.

Contents

List of Figures

List of Tables

1 Overview

This document provides information about the AndesCore NX25(F) processor, and associated platform/peripheral IPs that come with the NX25(F) release.

The organization of this document is as follows: the processor is described first, followed by descriptions regarding the associated AE350 platform in Section 17, and RISC-V specific platform IP components in Section 18, Section 19 and Section 20, followed by simulation guides in Section 23 and synthesis and FPGA guides starting from Section 26.

1.1 Features

The main features of the processor are:

CPU Core

- 5-stage in-order execution pipeline
- Hardware multiplier
  - radix-2/radix-4/radix-16/radix-256/fast
- Hardware divider
- Optional branch prediction
  - 4-entry return address stack (RAS)
  - Choice of
    - Static branch prediction, or
    - Dynamic branch prediction
      - 32/64/128/256-entry branch target buffer (BTB)
      - 256-entry branch history table
      - 8-bit global branch history
- Machine mode and optional User mode
- Optional performance monitors
- Misaligned memory accesses
- RISC-V physical memory protection

AndeStar V5 ISA

- RISC-V RV64I base integer instruction set
- RISC-V "C" standard extension for compressed instructions
- RISC-V "M" standard extension for integer multiplication and division
- Optional RISC-V "A" standard extension for atomic instructions
- Optional RISC-V "B" standard extension for bit manipulation

- Optional RISC-V “N” standard extension for user-level interrupt and exception handling
- Optional RISC-V “F” and “D” standard extensions for single/double-precision floating-point
- Andes Performance extension
- Andes CoDense extension

Andes Custom Extension

- Instruction encoding space up to 25 bits
- Concise Verilog description for RTL design
- Operands from existing GPR and memory
- Operands from ACE registers (ACR) and ACE memories (ACM)
- Single/multi-cycle instructions
- Vector instructions
- Background instructions
- Custom error status
- Block-level self-checking verification environment with standard Universal Verification Methodology (UVM)

Memory Subsystem

- I & D-Caches
  - Cache size: 4KiB/8KiB/16KiB/32KiB/64KiB
  - Cache line size: 32 bytes
  - Set associativity: Direct-mapped/2-way/4-way
  - Custom cache control operation through CSR read/write
- I & D local memories
  - Size: 4KiB to 16MiB
  - Optional local memory (LM) slave port
  - Interface: RAM or AHB-Lite
- Memory subsystem soft-error protection
  - Protection scheme: parity-checking or error-checking-and-correction (ECC)
  - Automatic hardware error correction
  - Protected memories:
    - I-Cache tag RAM and data RAM
    - D-Cache tag RAM and data RAM
    - I & D local memories

Bus

- Interface Protocol
  - Synchronous AHB, or
  - Synchronous AXI4

- 64-bit data width
- Configurable address width: 32–64 bits
- Low latency 1:1 clock ratio AHB access mode

Power Management

- Wait-for-interrupt (WFI) mode

Debug

- RISC-V External Debug Support
- Configurable number of breakpoints: 2/4/8
- External JTAG debug transport module
  - JTAG: IEEE Std 1149.1 style 4-wire JTAG interface
  - Serial: Andes 2-wire serial debug interface

Trace

- Optional instruction trace interface compliant to the ratified RISC-V Processor Trace Specification

AndeStar Extension

- StackSafe hardware stack protection extension
- PowerBrake simple power/performance scaling extension
- Custom performance counter events

Platform-Level Interrupt Controller (PLIC)

- Configurable number of interrupts: 1–1023
- Configurable number of interrupt priorities: 3/7/15/31/63/127/255
- Configurable number of targets: 1–16
- Andes Vectored Interrupt extension
- Configurable interrupt trigger types that are optionally programmable

1.2 Block Diagram

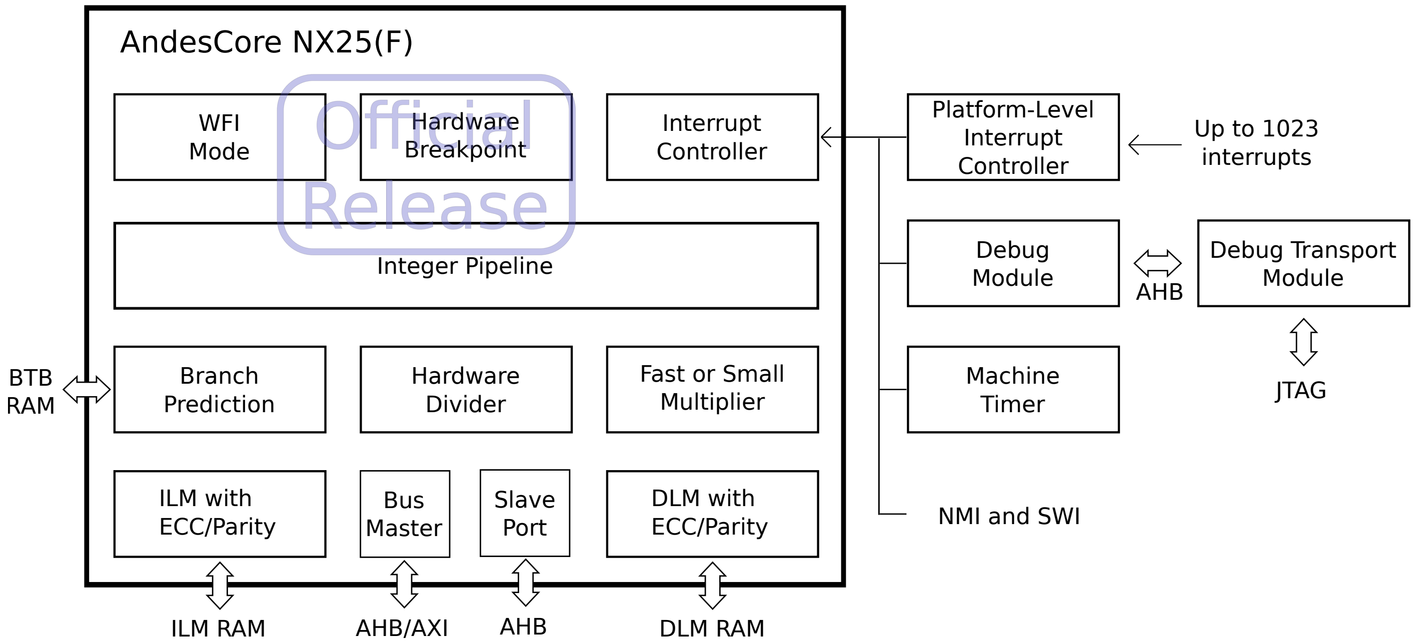


Figure 1: NX25(F) Block Diagram

1.2.1 Major Components

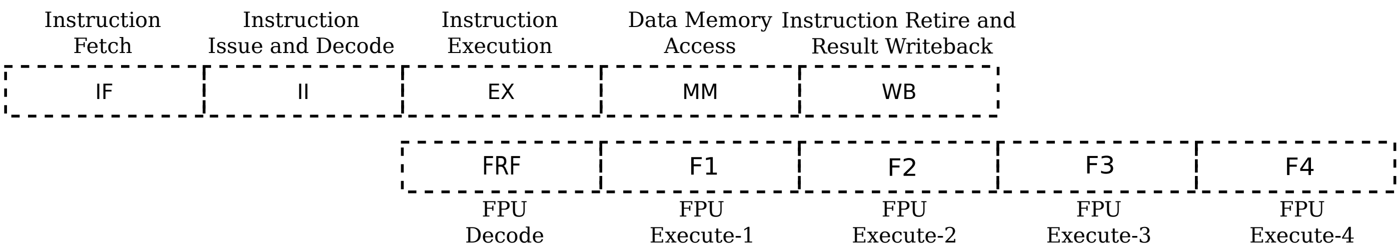
The following list describes the major components of the processor:

ACE Andes Custom Extension
ALU Arithmetic Logic Unit
BIU Bus Interface Unit
CSR Control and Status Register
DCU Data Cache Unit
DLM Data Local Memory Controller
FASTMUL Fast Multiplier
FPU Floating Point Unit
ICU Instruction Cache Unit
IFU Instruction Fetch Unit
ILM Instruction Local Memory Controller
IPIPE Integer Pipeline
LSU Load Store Unit
MDU Multiplication and Division Unit
PMP Physical Memory Protection Unit

RF	Register File
TRIGM	Trigger Module

1.3 Pipeline Stages and Activities

The processor implements a five-stage pipeline architecture. The following figure shows the pipeline stages.



The pipeline activities of the corresponding stages are:

IF—Instruction Fetch

- Fetching an instruction word from ILM/I-Cache/Bus
- Dynamic branch prediction

II—Instruction Decode and Issue

- 16/32-bit instruction alignment
- Instruction decoding
- Register file read
- Resolving data dependency
- Static branch prediction

EX—Instruction Execution

- ALU instruction execution
- Load/Store address generation

MM—Memory Access

- DLM/D-Cache access
- Division instruction execution
- Multiplication instruction execution
- Branch resolution

WB—Instruction Retire and Result Write-Back

- Interrupt resolution
- Instruction retire
- Register file write back
- ILM access
- Bus access
- ACE instruction execution

FRF—FPU Instruction Decode

- Instruction decoding
- Register file read

F1~F4—FPU Instruction Execution

- Floating-point arithmetic execution
- Data exchange between Integer/FPU pipelines

1.4 Design Hierarchy

The NX25(F) release package comes with the reference platform design, AE350. For processor-only licensing terms, the platform modules will be delivered encrypted as the testbench for the processor.

AE350 (see Section 17) supports two framework types: AHB and AXI. The design hierarchies are illustrated respectively in Figure 2 and Figure 4. The top-level module of this platform is ae350_chip. The ae350_chip module instantiates the NX25(F) processor. The top of the NX25(F) processor design is nx25_core.

nx25_core itself does not include any SRAM cells. All of the required SRAM cells are instantiated outside of nx25_core to make the design clean. Consequently, the nx25_core_top design is created to instantiate nx25_core along with all required SRAM cells.

The ae350_cpu_subsystem module is a reference subsystem that instantiates nx25_core_top and other tightly-coupled modules such as the debug subsystem. ae350_chip and ae350_cpu_subsystem are free for modification to meet system requirements.

Note

ae350_chip and ae350_cpu_subsystem modules will be overwritten when the configuration tool is re-run. Back up local changes before re-running the configuration tool.

1.4.1 AHB Framework

This framework type is chosen when the **Bus Type** option is configured as "ahb" in the configuration tool.

ncepldm200, nceplmt100, and nceplic100 modules are platform IPs as specified in the RISC-V architecture for proper operations of a RISC-V system. The ncepldm200 implements the debug functionality. nceplmt100 implements the RISC-V machine timer, and nceplic100 implements the RISC-V platform-level interrupt controller (PLIC).

The PLIC module is instantiated twice: `u_plic` for arbitrating interrupts from peripheral devices, and `u_plic_sw` for supporting software interrupts. The `u_plic_sw` instantiation only needs to use the programmability of the PLIC registers to generate (software programmable) interrupts, so all its interrupt sources are tied to zero.

atcbmc200, atcbusdec200, atcspi200, ae350_smu, atcrtc100, atcapbbrg100, atcwdt200, atciic100, atcpit100, atcuart100, atcgpio100, and some other necessary modules are pre-integrated Andes platform/peripheral IPs that may be shipped together with the NX25(F) package, depending on licensing agreements.

The ae350_bus_connector module merges the AHB bus and the related up-sizer and down-sizer. The block diagram is depicted in Figure 3

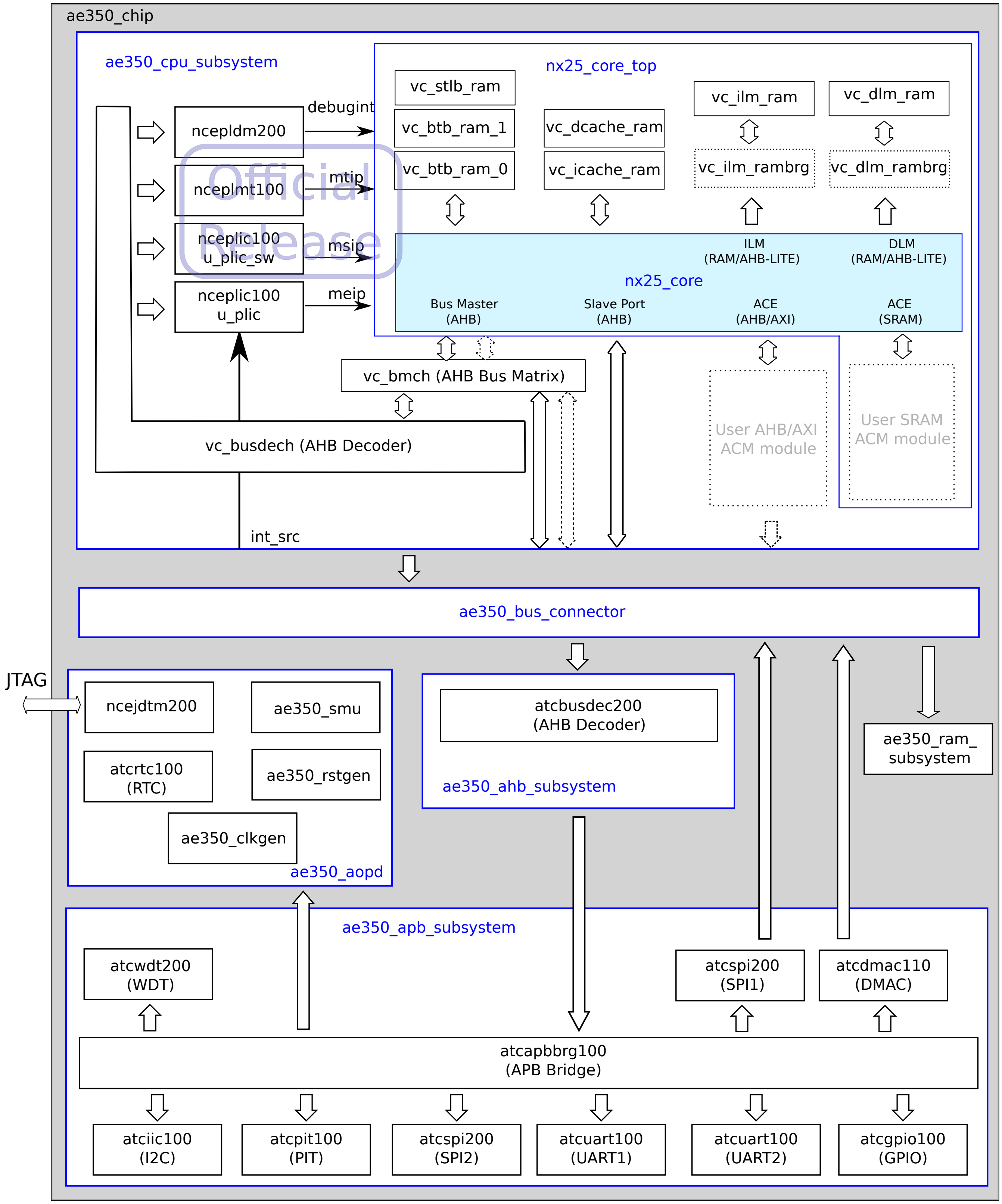


Figure 2: NX25(F) Design Hierarchy for the AHB Framework

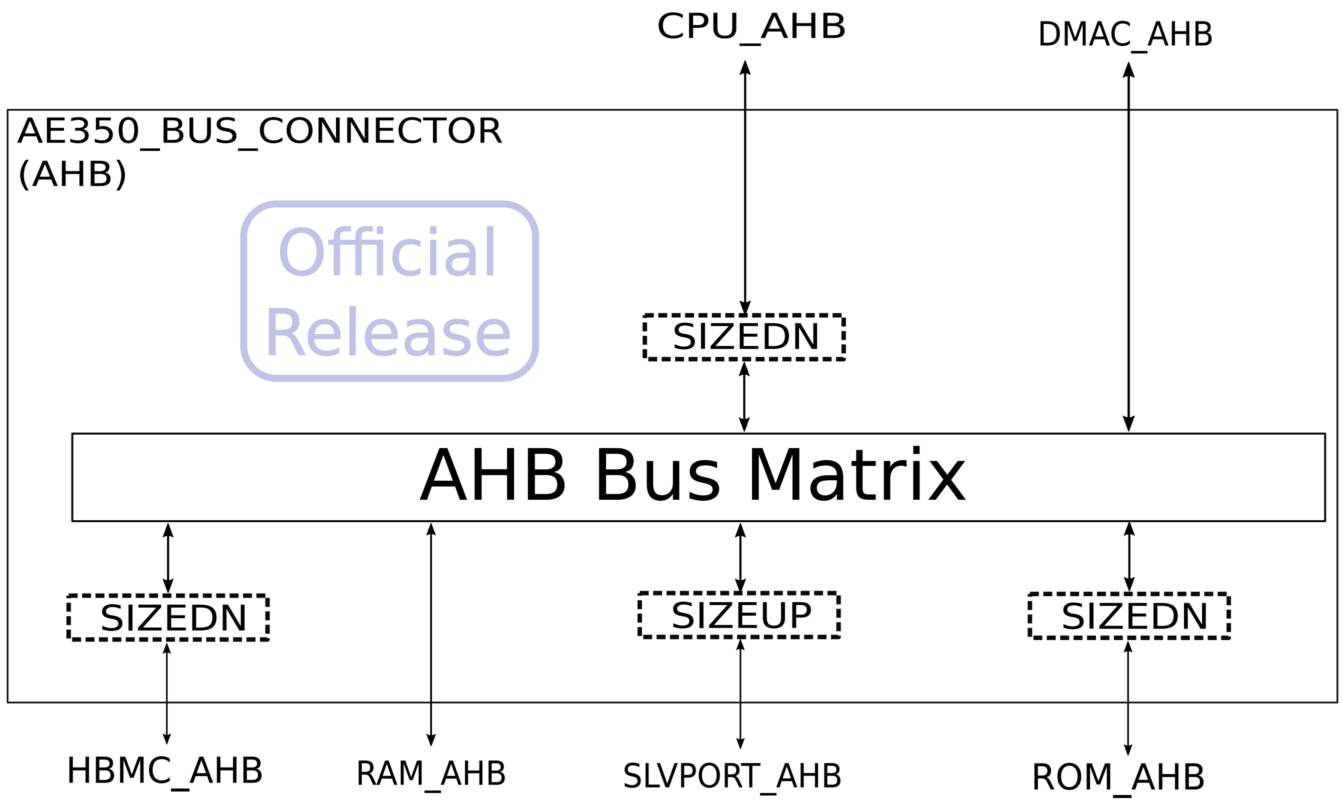


Figure 3: The AE350 Bus Connector for AHB Framework

1.4.2 AXI Framework

This framework type is chosen when the **Bus Type** option is configured as “axi” in the configuration tool.

The local memory slave port of NX25(F) only supports the AHB interface, so it is connected to the AXI bus matrix through an AXI-to-AHB bridge.

ncepldm200, nceplmt100, and nceplic100 modules are platform IPs as specified in the RISC-V architecture for proper operations of a RISC-V system. The ncepldm200 implements the debug functionality. nceplmt100 implements the RISC-V machine timer, and nceplic100 implements the RISC-V platform-level interrupt controller (PLIC).

The PLIC module is instantiated twice: `u_plic` for arbitrating interrupts from peripheral devices, and `u_plic_sw` for supporting software interrupts. The `u_plic_sw` instantiation only needs to use the programmability of the PLIC registers to generate (software programmable) interrupts, so all its interrupt sources are tied to zero.

atcbmc300, atcbusdec200, atcspi200, ae350_smu, atcrtc100, atcapbbrg100, atcwdt200, atciic100, atcpit100, atcuart100, atcgpio100, and some other necessary modules are pre-integrated Andes platform/peripheral IPs that may be shipped together with the NX25(F) package, depending on licensing agreements.

The ae350_bus_connector module merges the AXI bus and the related up-sizer, down-sizer, and AXI-AHB conversion bridge. The block diagram is depicted in Figure 5

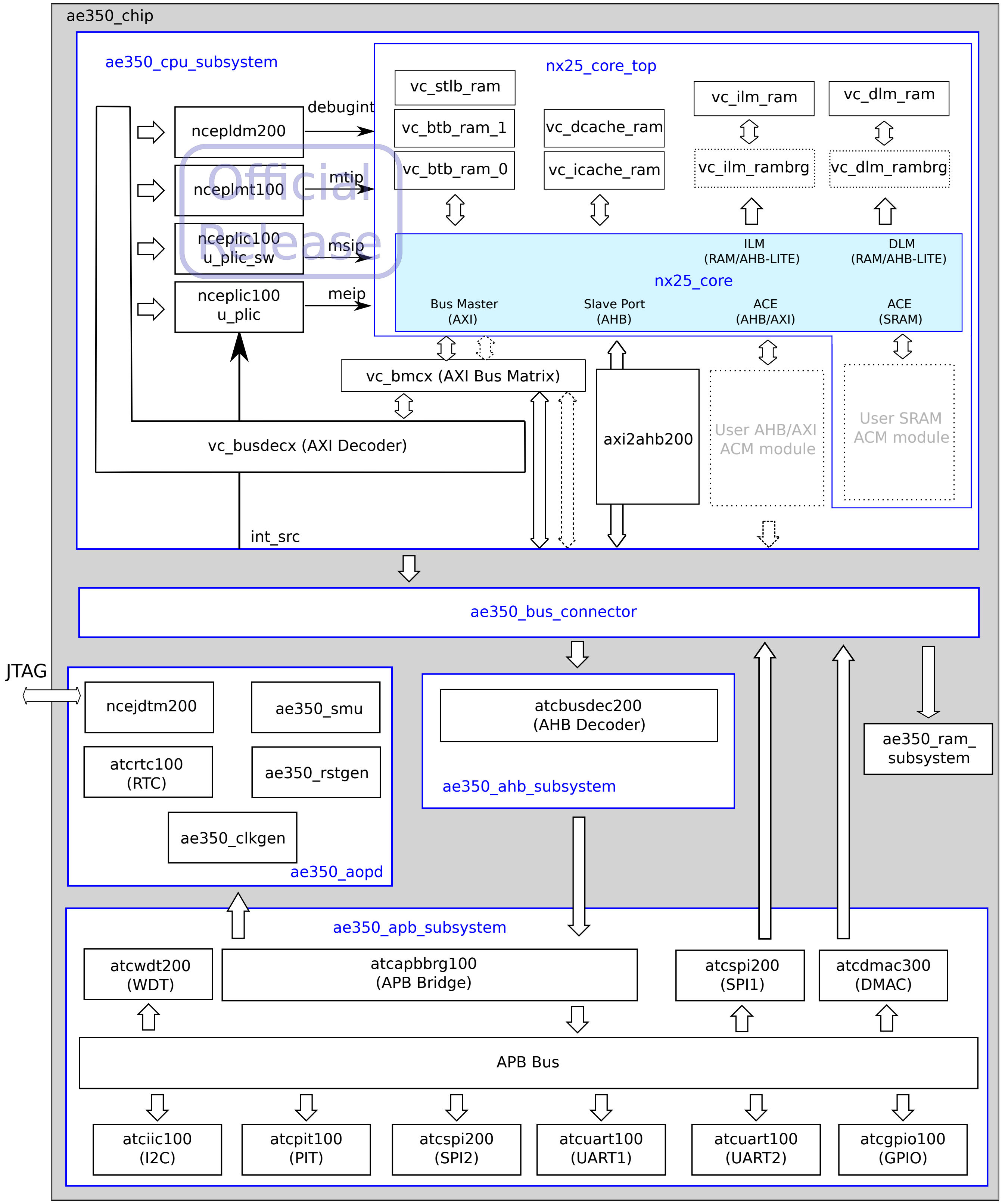


Figure 4: Design Hierarchy for the AXI Framework

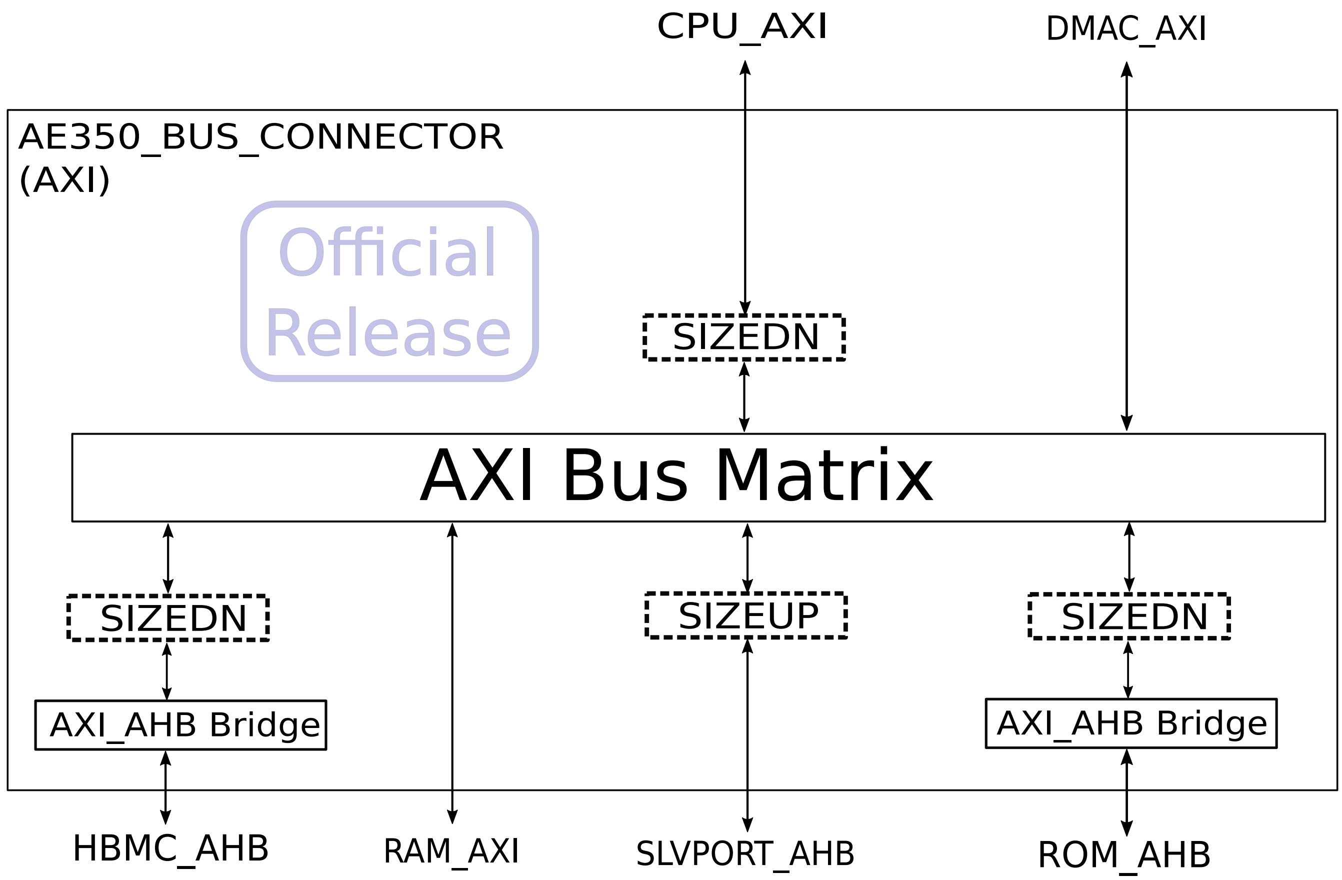


Figure 5: The AE350 Bus Connector for AXI Framework

1.5 Directory Structure

Unless otherwise stated, all directory paths in this document are relative to the environment variable **$NDS_HOME**, which points to the top of the distribution directory. The following table is a summary of directories in the NX25(F) distribution:

Table 1: Directory Structure

Directory	Description
$NDS_HOME/andes_ip	Design related files
$NDS_HOME/andes_ip/vc_core	NX25(F) core design
$NDS_HOME/andes_ip/ae350/top/hdl	AE350 design
$NDS_HOME/andes_ip/peripheral_ip	Peripheral IPs
$NDS_HOME/andes_ip/soc/nceplic100/hdl	PLIC design
$NDS_HOME/andes_ip/soc/nceplmt100/hdl	External machine timer design
$NDS_HOME/andes_ip/soc/ncepldm200/hdl	External debug module design

Continued on next page...

Table 1: (continued)

Directory	Description
$NDS_HOME/andes_ip/soc/ncejdtm200/hdl	External JTAG debug transport module design
$NDS_HOME/testbench	Testbench related files — top-level modules
$NDS_HOME/andes_vip/models	Directory for simulator models — the external memory model, the external debug host model and AHB/AXI slave models.
$NDS_HOME/andes_vip/patterns/samples	Sample test patterns. See Section 23.3.8.
$NDS_HOME/andes_vip/patterns/riscv-tests	RISC-V test patterns
$NDS_HOME/flists	Directory for simulation file lists
$NDS_HOME/tools/bin	Tools for simulation
$NDS_HOME/config_tools	Configuration tools

2 Processor Configuration Options

The NX25(F) processor is configured through the bundled configuration tool.

2.1 Configuration Tool

The configuration tool allows interactive selection of configurable options. The tool is Tcl/Tk based and it requires the Tk interpreter `wish` to exist in the search path. To start the tool, make sure that the **$DISPLAY** environment variable is set correctly to point to a valid X server, and then type the following command to launch the configuration tool.

$NDS_HOME/config_tools/nds-softcore-config

Figure 6 shows a screenshot of the tool. The "Save" button saves your options so that it could later be loaded into the tool. The "Generate nx25_core" button configures the NX25(F) processor with the selected options. By clicking this button, the following files are generated and overwritten. If required, back up the files before re-configuring the processor.

- **$NDS_HOME**/andes_ip/vc_core/top/hdl/nx25_core.v
  - Top module of the NX25(F) processor with the selected options.
- **$NDS_HOME**/andes_ip/vc_core/top/hdl/nx25_core_top.v
  - Top module of the NX25(F) processor with the selected options and all required SRAM cells.
- **$NDS_HOME**/andes_ip/vc_core/top/hdl/vc_core.v
  - Core design of the NX25(F) processor.
- **$NDS_HOME**/andes_ip/vc_core/top/hdl/ae350_cpu_subsystem.v
  - Sample CPU subsystem that instantiates the NX25(F) processor. This subsystem is for the AE350 platform.
- **$NDS_HOME**/andes_ip/vc_core/top/hdl/config.inc
  - Required by the accompanying testbench.
- **$NDS_HOME**/andes_ip/ae350/top/hdl/include/ae350_config.vh
  - Configurations for the platform, required by the AE350 CPU subsystem and chip.
- **$NDS_HOME**/andes_ip/vc_core/top/hdl/nx25_core.xml
  - IP-XACT XML for the NX25(F) processor.

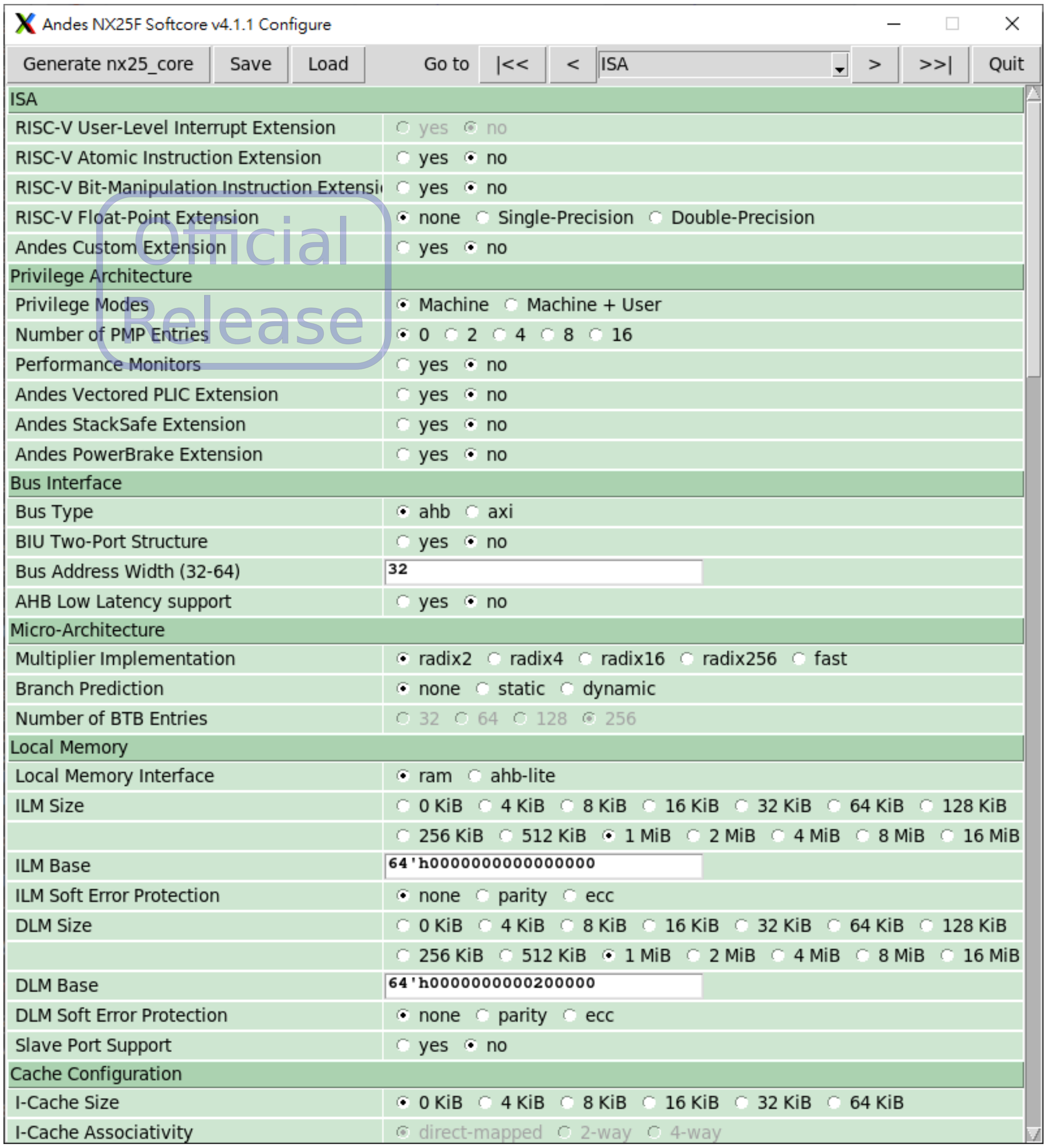


Figure 6: `nds-softcore-config` Screenshot

Note

The screenshot only serves to show how the tool *looks like* to facilitate the description above. It is not a goal for this figure to show the most updated version of the tool. The actual content of the tool may differ slightly as the tool gets updated with new features added to NX25(F). Please see Table 2 for the most updated list of all configurable options.

2.2 Summary of Configuration

Table 2: NX25(F) Configuration Options

Option	Valid Values	Description
ISA		
RISC-V User-Level Interrupt Extension	yes / no	Section 2.3.1
RISC-V Atomic Instruction Extension	yes / no	Section 2.3.2
RISC-V Floating-Point Instruction Extension	double+single precision / single precision / none	Section 2.3.3
RISC-V Bit-Manipulation Instruction Extension	yes / no	Section 2.3.4
Andes Custom Extension (ACE)	yes / no	Section 2.3.5
Privilege Architecture		
Privilege Modes	Machine / Machine + User	Section 2.4.1
Number of PMP Entries	0 / 2 / 4 / 8 / 16	Section 2.4.2
Performance Monitors	yes / no	Section 2.4.3
Andes Vectored PLIC Extension	yes / no	Section 2.4.4
Andes StackSafe Extension	yes / no	Section 2.4.5
Andes PowerBrake Extension	yes / no	Section 2.4.6
Bus Interface		
Bus Type	ahb / axi	Section 2.5
Low Access-Latency AHB Bus	yes / no	Section 2.5
Bus Address Width	32-64	Section 2.5
BIU Two-Port Structure	yes / no	Section 2.5
Micro-Architecture		
Multiplier Implementation	radix2 / radix4 / radix16 / radix256 / fast	Section 2.6.1
Branch Prediction	none / static / dynamic	Section 2.6.2
Number of BTB Entries	32 / 64 / 128 / 256	Section 2.6.2
Local Memory		
Local Memory Interface	ram / ahb-lite	Section 2.7.1
ILM Size	0 KiB / 4 KiB / 8 KiB / 16 KiB / 32 KiB / 64 KiB / 128 KiB / 256 KiB / 512 KiB / 1 MiB / 2 MiB / 4 MiB / 8 MiB / 16 MiB	Section 2.7.2
ILM Base		Section 2.7.2
ILM Soft Error Protection	none / parity / ecc	Section 2.7.2
DLM Size	0 KiB / 4 KiB / 8 KiB / 16 KiB / 32 KiB / 64 KiB / 128 KiB / 256 KiB / 512 KiB / 1 MiB / 2 MiB / 4 MiB / 8 MiB / 16 MiB	Section 2.7.3

Continued on next page...

Table 2: (continued)

Option	Valid Values	Description
DLM Base		Section 2.7.3
DLM Soft Error Protection	none / parity / ecc	Section 2.7.3
Slave Port Support	yes / no	Section 2.7.4
Cache Configuration		
I-Cache Size	0 KiB / 4 KiB / 8 KiB / 16 KiB / 32 KiB / 64 KiB	Section 2.8.1
I-Cache Associativity	direct-mapped / 2-way / 4-way	Section 2.8.1
I-Cache Replacement Policy	random / pseudo-lru	Section 2.8.1
I-Cache Soft Error Protection	none / parity / ecc	Section 2.8.1
I-Cache Cache-Line Filling Policy	First-Word-First / Critical-Word-First	Section 2.8.1
D-Cache Size	0 KiB / 4 KiB / 8 KiB / 16 KiB / 32 KiB / 64 KiB	Section 2.8.2
D-Cache Associativity	direct-mapped / 2-way / 4-way	Section 2.8.2
D-Cache Replacement Policy	random/pseudo-lru	Section 2.8.2
D-Cache Soft Error Protection	none / parity / ecc	Section 2.8.2
Debug and Trace		
Debug Support	yes / no	Section 2.9.1
Debug Module Base		Section 2.9.2
Number of Triggers	2 / 4 / 8	Section 2.9.3
Trace Interface	none / instruction	Section 2.9.4
Platform Debug Options		
Debug Interface	jtag / serial	Section 2.10.1
PLDM System Bus Access	yes / no	Section 2.10.2
PLDM Program Buffer	1 / 2 / 8	Section 2.10.3
PLDM Group Halting	0 / 1 / 2 / 3	Section 2.10.4
Clock Domain Crossing		
Synchronizer Level	2 / 3	Section 2.11.1
Device Region		
Device Region*N* Base		Section 2.12
Device Region*N* Mask		Section 2.12
Writethrough Region		
Writethrough Region*N* Base		Section 2.13
Writethrough Region*N* Mask		Section 2.13
Platform Peripheral IP		

Continued on next page. . .

Table 2: (continued)

Option	Valid Values	Description
DMA Support	yes / no	Section 2.14.1
GPIO Support	yes / no	Section 2.14.2
I2C Support	yes / no	Section 2.14.3
PIT Support	yes / no	Section 2.14.4
RTC Support	yes / no	Section 2.14.5
SPI1 Support	yes / no	Section 2.14.6
SPI2 Support	yes / no	Section 2.14.6
UART1 Support	yes / no	Section 2.14.7
UART2 Support	yes / no	Section 2.14.7
WDT Support	yes / no	Section 2.14.8

2.3 ISA

2.3.1 RISC-V User-Level Interrupt Extension

Specifying this option to "yes" enables the RISC-V User-Level Interrupt Extension (RVN).

2.3.2 RISC-V Atomic Instruction Extension

Specifying this option to "yes" enables the RISC-V "A" Standard Extension for Atomic Instructions (RVA). RVA includes instructions that atomically read-modify-write memories for synchronization between multiple hardware threads (harts).

2.3.3 RISC-V Floating-Point Instruction Extension

This option determines the floating-point extension type.

- none: no floating-point extension
- single precision: "F" standard extension for single-precision floating-point instruction
- double+single precision: "F" and "D" standard extensions for single- and double-precision floating-point instructions

Note

The availability of the floating-point extension depends on NX25(F) licensing agreements.

2.3.4 RISC-V Bit-Manipulation ISA

Specifying this option to "yes" enables the RISC-V "B" Standard Extension for bit-manipulation instructions.

2.3.5 Andes Custom Extension

Specifying this option to "yes" enables the custom instruction extension.

The Andes Custom Extension (ACE) feature provides a simple and flexible interface to extend new instructions. Please see *Andes Custom Extension Specification* and *Andes Custom Extension User Manual* for more information.

Note

The ACE feature requires additional licenses. Please contact Andes Technology for further information.

2.4 Privilege Architecture

2.4.1 Privilege Modes

This option determines the number of supported privileges levels. The NX25(F) processor supports 1 or 2 privilege levels, as shown in Table 3.

Table 3: Supported Combinations of Privilege Modes

Number of Levels	Supported Modes	Intended Usage
1	Machine	Simple embedded systems
2	Machine, User	Secure embedded systems

The RISC-V privilege architecture specifies four privilege levels as shown in Table 4. Machine mode (M-mode) has the highest privileges and is the mandatory privilege level for a RISC-V hardware platform. User mode (U-mode) restricts privileges to protect against incorrect or malicious application codes. And Supervisor mode (S-mode) is provided for Unix-like operating systems with address translating and protection requirements.

Table 4: RISC-V Privilege Levels

Level	Encoding	Name	Abbreviation
0	00	User/Application	U

Continued on next page...

Table 4: (continued)

Level	Encoding	Name	Abbreviation
1	01	Supervisor	S
2	10	Reserved	
3	11	Machine	M

2.4.2 Number of Physical Memory Protection Entries

This option selects the number of supported PMP entries.

2.4.3 Performance Monitors

Specifying this option to "yes" enables hardware performance monitors, including `mcycle` and `minstret` Control and Status Registers (CSR). The `mcycle` register counts the elapsed clock cycles while the `minstret` register counts the number of retired instructions.

2.4.4 Andes Vectored PLIC Extension

Specifying this option to "yes" enables the Andes Vectored PLIC (VPLIC) extension for reducing interrupt latency. See Section 18.3 for more information.

2.4.5 Andes StackSafe Extension

Specifying this option to "yes" enables the StackSafe hardware stack protection extension.

The extension is a hardware mechanism for tracking and guarding against the stack pointer overflows/underflows. See Section 15.11.1 for more information.

2.4.6 Andes PowerBrake Extension

Specifying this option to "yes" enables the PowerBrake power/performance scaling extension.

The PowerBrake extension throttles performance by reducing instruction executing rate instead of slowing down clock frequency. The performance and hence power consumption can be switched to a different level in a couple of instructions. This is an ultra-low latency mechanism for performance & power scaling, as compared to the latencies of frequency scaling through PLL programming.

2.5 Bus Interface

The bus interface unit (BIU) is responsible for system bus accesses of NX25(F). It has several configuration options as mentioned below.

The **Bus Type** option determines the interface type used by the CPU core and also the framework type of the associated platform.

The **Bus Data Width** determines the value of Verilog parameter `BIU_DATA_WIDTH`. It is not configurable and is fixed at 64 bits wide.

The **Bus Address Width** option determines the value of Verilog parameter `BIU_ADDR_WIDTH`.

The width of the bus address could be any value between 32 and 64 bits.

Note that the reference Andes AXI platform requires **Bus Address Width** to be 33 bits. Please see Table 102 for details.

The **Low Access-Latency AHB Bus** option reduces the system bus access latency at the expense of the maximum frequency, which drops by around 30% compared to those of configurations without this option. This option is expected to be used under configurations where there are no caches.

The **BIU Two-Port Structure** option splits instruction and data accesses to separate ports. For detailed usage and restrictions, please see Section 10.3.

2.6 Micro Architecture

2.6.1 Multiplier Implementation

This option selects the implementation of the hardware multiplier in NX25(F). Valid values and the respective performance are:

- radix2: 1-bit/cycle
- radix4: 2-bit/cycle
- radix16: 4-bit/cycle
- radix256: 8-bit/cycle
- fast: two-stage pipelined.

Radix multiplier types realize the multiplier through serial additions, while the fast multiplier type implements a two-stage pipelined parallel multiplier.

2.6.2 Branch Prediction

This option selects the branch prediction scheme: static or dynamic branch prediction.

The static branch prediction algorithm predicts that all backward branches will be taken and all forward branches will not. The dynamic branch prediction algorithm supports a configurable number of BTB entries and uses a 2-way branch target buffer (BTB) along with a branch history table to predict the target address and direction (taken/not-taken) of each branch.

The "static" branch prediction scheme implements the static branch algorithm. However, the "dynamic" branch prediction implements both static and dynamic prediction algorithms to improve the prediction efficiency.

In addition, when either "static" or "dynamic" branch prediction is selected, a 4-entry return address stack (RAS) will also be instantiated for predicting the return addresses of function calls.

Note that branch prediction algorithms may cause instruction fetch to speculatively go out of the code (text) region. To avoid unexpected system issues with speculative fetches, the following requirements should be met to enable branch prediction:

- All bus addresses must be able to return responses.
- PMP (or other system-level protection logic) should be used to prevent speculative fetch from accessing regions with side effects.

If a platform cannot meet the two conditions above, branch prediction could cause system hang when privilege mode (virtual address translation) changes, or unexpected side effects to device regions.

2.7 Local Memory

2.7.1 Local Memory Interface

NX25(F) supports two local memory interfaces:

- ram: for connecting to memory devices without wait states
- ahb-lite: for connecting to AHB-lite slaves.

Please note that local memory soft-error protection scheme and the **Slave Port Support** option are only available when the interface type is "ram".

2.7.2 Instruction Local Memory (ILM)

The **ILM Size** option (ILM_SIZE_KB) selects the size of the instruction local memory in KiB. Only power-of-2 sizes are supported. Specifying 0 to this option unconfigures the instruction local memory.

The **ILM Base** option (ILM_BASE) specifies the base address of the instruction local memory. The address must align to the size of the instruction local memory, and the effective address width should not exceed physical address width .

The **ILM Soft Error Protection** option selects the soft-error protection scheme for the instruction local memory:

- none: no protection
- parity: single-error detection
- ecc: single-error correction, and double-error detection

2.7.3 Data Local Memory (DLM)

The **DLM Size** option (DLM_SIZE_KB) selects the size of the data local memory in KiB. Only power-of-2 sizes are supported. Specifying 0 to this option unconfigures the data local memory.

The **DLM Base** option (DLM_BASE) specifies the base address of the data local memory. The address must align to the size of the data local memory, and the effective address width should not exceed physical address width .

The **DLM Soft Error Protection** option selects the soft-error protection scheme for the data local memory:

- none: no protection
- parity: single-error detection
- ecc: single-error correction, and double-error detection

2.7.4 Slave Port Support

Specifying this option to “yes” enables the local memory slave port for bus masters to access the local memories of NX25(F).

The interface protocol for the local memory slave port is AHB. The data width (`SLAVE_PORT_DATA_WIDTH`) is identical to bus data width (`BIU_DATA_WIDTH`). Please see Section 8 for details of local memory slave ports.

2.8 Cache Configuration

2.8.1 Instruction Cache

The **I-Cache Size** option selects the size of the instruction cache in KiB. Only power-of-2 sizes are supported. Specifying 0 to this option unconfigures the instruction cache.

The **I-Cache Associativity** option selects the associativity of instruction cache. But note that the 4KiB size configuration only supports "direct-mapped".

The **I-Cache Replacement Policy** option selects the replacement policy of instruction cache on cache misses.

The **I-Cache Soft Error Protection** option selects the soft-error protection scheme for the instruction cache:

- none: no protection
- parity: single-error correction *
- ecc: single-error and double-error correction *

Note

- The "correction" is performed by re-fetching the clean instruction from the next-level memory.

The **I-Cache Cache-Line Filling Policy** option selects the filling policy for the I-Cache cache line.

2.8.2 Data Cache

The **D-Cache Size** option selects the size of the data cache in KiB. Only power-of-2 sizes are supported. Specifying 0 to this option unconfigures the data cache.

The **D-Cache Associativity** option selects the associativity of data cache. But note that the 4KiB size configuration only supports "direct-mapped".

The **D-Cache Replacement Policy** option selects the replacement policy of data cache when cache miss.

The **D-Cache Soft Error Protection** option selects the soft-error protection scheme for the data cache:

- none: no protection
- parity: single-error detection
- ecc: single-error correction, and double-error detection

2.9 Debug and Trace

2.9.1 Debug Support

Specifying this option to “yes” enables the core debug support.

2.9.2 Debug Module Base

The **Debug Module Base** option specifies the entry point address of the exception handler for servicing debug exceptions in the debug mode. It should be set to the address of the Debug ROM of the NCEPLDM200 debug module, which is also the base address of the module. This address space should be a *device region* for proper operations of the external debug support. See Section 20.5 for more information.

2.9.3 Number of Trigger

The **Number of Trigger** option selects the number of hardware breakpoints.

2.9.4 Trace Interface

The **Trace Interface** option specifies the type of trace interface. This interface can be set to one of the following options:

- **none**: No trace support
- **instruction**: Interface compliant to the ratified RISC-V Processor Trace Specification

2.10 Platform Debug Options

This group of options is used to configure the platform debug subsystem, which consists of JTAG Debug Transfer Module (JDTM), Platform Debug Module (PLDM) and the connections between these modules. Unlike other configuration options, these options will be written to the platform configuration file instead of to the core configuration file, and they will also affect the platform designs.

When the core debug feature (Section 2.9.1) is disabled, the debug subsystem and the related I/O ports will be removed, and all the options in this group will be disabled.

2.10.1 Debug Interface

The **Debug Interface** option specifies the interface between JDTM and the external device for debugging. This interface can be set to one of the following two types:

- **jtag**: Standard JTAG interface
- **serial**: Andes 2-wire serial debug interface

This option will affect platform I/O ports and I/O related modules.

2.10.2 PLDM System Bus Access

Specifying this option to "yes" will allow PLDM to directly access the system bus by integrating a bus master into PLDM; otherwise, the CPU core will be utilized to access the system bus.

2.10.3 PLDM Program Buffer

This option specifies the size of program buffer embedded in PLDM, in 32-bit words. Program buffer is a set of registers which can be programmed through the debug interface and accessed by the target hart.

2.10.4 PLDM Group Halting

This option specifies how many non-default halt groups are supported by PLDM.

Each hart can be assigned to a halt group. When one hart in a non-default halt group is halted, all the other harts in the same group will also be halted as soon as possible.

For more details about the group halting feature, please see Section 20.5.22.

2.11 Clock Domain Crossing

2.11.1 Synchronizer Level

This option specifies the level N ($N = 2$ or 3) of the CDC synchronizer.

The CDC synchronizer is designed to avoid metastability caused clock domain crossing. Increasing the level of the CDC synchronizers may reduce the risk of metastability.

All the signals on the core and the platform that cross clock domains are affected by this option.

2.12 Device Region

NX25(F) can specify at most sixteen static device regions, where a device region N ($N = 0 – 15$) is defined by the following two configuration options:

- **Device Region*N* Base**: the base address of the region,
- **Device Region*N* Mask**: the address mask of the region.

An address is inside a region when:

> (address[PALEN-1:12] & **Device Region*N* Mask[PALEN-1:12]**) == **Device Region*N* Base[PALEN-1:12]**.

A device region can be disabled by setting **Device Region*N* Base** to all ones and **Device Region*N* Mask** to all zeros. The minimum region size is 4 KiB.

Note that PMA entries have higher priorities than the static Device Region and Write-through Region settings. Please see Section 6 for details about physical memory attributes, ordering and device regions.

2.13 Writethrough Region

NX25(F) can specify at most sixteen write-through regions, where a write-through region N ($N = 0 –15$) is defined by two configuration options:

- **Writethrough Region*N* Base**: the base address of the region,
- **Writethrough Region*N* Mask**: the address mask of the region.

An address is inside a region when:

> (address[PALEN-1:12] & **Writethrough Region*N* Mask**[PALEN-1:12]) == **Writethrough Region*N* Base**[PALEN-1:12].

A write-through region can be disabled by setting **Writethrough Region*N* Base** to all ones and **Writethrough Region*N* Mask** to all zeros. The minimum region size is 4 KiB.

Please see Section 6 for details about write-through regions.

2.14 Platform Peripheral IP

The configuration options in this section control whether the corresponding peripheral IP should be instantiated in the AE350 platform.

2.14.1 DMA Support

Specifying this option to "yes" instantiates the DMA (Direct-Memory-Access) controller in the platform.

If the bus interface type is specified to "ahb", ATCDMAC110 will be used. Otherwise, ATCDMAC300 will be used when the AXI interface is selected.

See Section 17.7 for more details about the DMA controller.

2.14.2 GPIO Support

Specifying this option to "yes" instantiates the GPIO controller in the platform. Note that two seven-segment LEDs and buttons are connected to GPIO ports on the FPGA board. If the GPIO support is not configured, these LEDs and buttons will not be accessible. See Section 26.1.5 for GPIO pin assignments on the FPGA board, and Section 17.7 for more details about the GPIO controller.

2.14.3 I2C Support

Specifying this option to "yes" instantiates the I2C controller in the platform. Note that the I2C ROM is connected to the I2C port on the FPGA board. If the I2C support is not configured, the I2C ROM will not be accessible. See Section 26.1.5 for I2C pin assignments on the FPGA board, and Section 17.7 for more details about the I2C controller.

2.14.4 PIT Support

Specifying this option to "yes" instantiates the PIT (Programmable Interval Timer) controller in the platform. See Section 17.7 for more details about the PIT controller.

2.14.5 RTC Support

Specifying this option to "yes" instantiates the RTC (Real-Time Clock) controller in the platform.

See Section 17.7 for more details about the RTC controller.

2.14.6 SPI Support

Specifying this option to "yes" instantiates the first (SPI1) or second (SPI2) SPI controller individually in the platform. Please note that the reset vector points to the SPI1 interface in the AE350 platform. The CPU core will fetch instructions from the ROM through SPI1 when the CPU core boots up. If SPI1 is not configured, it is necessary to point the reset vector to another interface.

See Section 17.7 for more details about the SPI controller.

2.14.7 UART Support

Specifying this option to "yes" instantiates the first (UART1) or second (UART2) UART controller individually in the platform.

UART2 is the default COM port on the FPGA board. If UART2 is not configured, CPU cannot be accessed through terminal emulators.

See Section 26.1.1 for UART pin assignments on the FPGA board, and Section 17.7 for more details about the UART controller.

2.14.8 WDT Support

Specifying this option to "yes" instantiates the WDT (WatchDog Timer) controller in the platform.

See Section 17.7 for more details about the WDT controller.

3 Signal Descriptions

This section describes the interface ports of the NX25(F) core. All signals are Active-High unless otherwise indicated.

3.1 General Signals

Table 5: General Signals

Signal Name	Direction	Description
hart_id[63:0]	input	This signal indicates the CPU hart ID. Hart IDs might not necessarily be numbered contiguously in a multiprocessor system, but at least one hart must have a hart ID of zero.
core_reset_n	input	CPU reset (Active-Low)
slv_reset_n	input	Reset signal (Active-Low) for the local memory slave port
test_mode	input	Scan test mode. Internal synchronized reset signals are disabled when this signal is asserted.
test_rstn	input	Reset signal (Active-Low) for test mode
core_clk	input	CPU clock input
dc_clk	input	D-Cache clock input. `dc_clk` should be tied to `core_clk`.
lm_clk	input	Local memory clock input. `lm_clk` and `core_clk` are synchronous but their gating conditions are different.
reset_vector[VALEN-1:0]	input	Default program counter value upon reset. It should normally be a 4-byte aligned value but 2-byte aligned value is also allowed. Bit 0 of this input signal should be zero.
bus_clk_en	input	This is a `core_clk` clock domain signal indicating that the data from the bus clock domain can be sampled at the coming rising edge of `core_clk`. See Section 4.2 for details.
core_wfi_mode	output	This signal indicates that the processor is in the wait-for-interrupt mode. See Section 13.1 for details.

3.2 Interrupt Signals

NX25(F) assumes the clock domain of interrupt signals is different from the `core_clk` clock domain. N ($N = 2$ or 3, See Section 2.11.1) stages of synchronization flip-flops are used to avoid metastability in an asynchronous clock crossing domain.

Table 6: Interrupt Signals

Signal Name	Direction	Description
meip	input	External interrupt pending
meiid[9:0]	input	External interrupt source ID, used in the vector interrupt mode
meiack	output	External interrupt acknowledgment, used in the vector interrupt mode
mtip	input	Timer interrupt pending
msip	input	Software interrupt pending
nmi	input	Non-maskable interrupt
ueip	input	User-mode external interrupt pending (present only when the RVN support is configured)
ueiid[9:0]	input	User-mode external interrupt source ID, used in the vector interrupt mode (present only when the RVN support is configured)
ueiack	output	User-mode external interrupt acknowledgment, used in the vector interrupt mode (present only when the RVN support is configured)

3.3 Debug Signals

NX25(F) assumes the clock domain of input signals `debugint` and `resethaltreq` are different from `core_clk`. N ($N = 2$ or 3, See Section 2.11.1) stages of synchronization flip-flops are used to avoid metastability in an asynchronous clock crossing domain.

Table 7: External Debug Signals

Signal Name	Direction	Description
debugint	input	Debug interrupt
resethaltreq	input	Halt-on-reset request

Continued on next page...

Table 7: (continued)

Signal Name	Direction	Description
hart_unavail	output	This signal indicates that the processor is not available for accesses by the external debugger. The processor may be in the reset or some kind of power-gating state.
hart_halted	output	This signal indicates that the processor is halted.
hart_under_reset	output	This signal indicates that the processor is under reset.
stoptime	output	This signal indicates that the processor is in Debug Mode and timers should stop counting. This signal is controlled by `dcsr.STOPTIME`.

3.4 Trace Signals

Table 8: Trace Signals for Ratified RISC-V Processor Trace

Signal Name	Direction	Description
trace_enabled	input	This signal indicates the trace interface is enabled; this interface will not be active if this signal is not set.
trace_itype[3:0]	output	This signal indicates the termination type of the instruction block. An instruction block contains all the instructions retired in a cycle.
trace_cause[9:0]	output	This signal indicates the cause of an exception or an interrupt.
trace_tval[63:0]	output	This signal indicates the associated trap value.
trace_priv[1:0]	output	This signal indicates the privilege level of all instructions retired in this cycle.
trace_iaddr[BIU_ADDR_WIDTH-1:0]	output	This signal indicates the address of the first instruction retired in this block.
trace_iretire[1:0]	output	This signal indicates the number of halfwords represented by the instructions retired in this block.
trace_ilastsize	output	This signal indicates that the size of the last retired instruction is $2^{ilastsize}$ half-words.

Continued on next page...

Table 8: (continued)

Signal Name	Direction	Description
trace_trigger[2:0]	output	This signal indicates the trigger events. A pulse on bit 0 will cause the encoder to start tracing. A pulse on bit 1 will cause the encoder to stop tracing. A pulse on bit 2 is a trace notify event.
trace_halted	output	This signal indicates that the hart is halted.
trace_reset	output	This signal indicates that the hart is under reset.

Note

For the ratified RISC-V processor Trace, the instruction trace interface implements the Multiple Retirement scheme. To connect to trace encoders supporting the Single Retirement scheme, all bits of the `trace_iretire` signal could be OR-together to reduce to a 1-bit signal.

3.5 AHB Interface Signals

AHB interface signals provide connectivity to the AHB system bus. These signals are used only when configuration option **Bus Interface — Bus Type** is "ahb" and **Bus Interface — BIU Two-port Structure** is "no" . Otherwise, they should be left unconnected. They are sampled/driven in the `core_clk` clock domain when `bus_clk_en` is HIGH. Please see Section 4.2 if the bus clock frequency needs to be lower than the `core_clk` frequency.

Table 9: AHB Interface Signals

Signal Name	Direction	Description
hgrant	input	Bus grant
hrdata[63:0]	input	Read data bus
hready	input	Transfer done
hresp[1:0]	input	Transfer response
haddr[BIU_ADDR_WIDTH-1:0]	output	Address bus
hburst[2:0]	output	Burst type
hbusreq	output	Bus request
hlock	output	Locked transfer
hprot[3:0]	output	Protection control
hsize[2:0]	output	Transfer size
htrans[1:0]	output	Transfer type

Continued on next page...

Table 9: (continued)

Signal Name	Direction	Description
hwdata[63:0]	output	Write data bus
hwrite	output	Transfer direction

3.6 AHBx2 Interface Signals

AHBx2 interface signals provide connectivity separately to the AHB instruction and data buses. These signals are used only when configuration option **Bus Interface — Bus Type** is "ahb" and **Bus Interface — BIU Two-port Structure** is "yes". Otherwise, they should be left unconnected. They are sampled/driven in the `core_clk` clock domain when `bus_clk_en` is HIGH. Please see Section 4.2 if the bus clock frequency needs to be lower than the `core_clk` frequency.

Table 10: AHBx2 Interface Signals

Signal Name	Direction	Description
i_hgrant	input	Instruction bus bus grant
i_hrdata[63:0]	input	Instruction bus read data bus
i_hready	input	Instruction bus transfer done
i_hresp[1:0]	input	Instruction bus transfer response
i_haddr[BIU_ADDR_WIDTH-1:0]	output	Instruction bus address bus
i_hburst[2:0]	output	Instruction bus burst type
i_hbusreq	output	Instruction bus bus request
i_hlock	output	Instruction bus locked transfer
i_hprot[3:0]	output	Instruction bus protection control
i_hsize[2:0]	output	Instruction bus transfer size
i_htrans[1:0]	output	Instruction bus transfer type
i_hwdata[63:0]	output	Instruction bus write data bus
i_hwrite	output	Instruction bus transfer direction
d_hgrant	input	Data bus bus grant
d_hrdata[63:0]	input	Data bus read data bus
d_hready	input	Data bus transfer done
d_hresp[1:0]	input	Data bus transfer response
d_haddr[BIU_ADDR_WIDTH-1:0]	output	Data bus address bus
d_hburst[2:0]	output	Data bus burst type
d_hbusreq	output	Data bus bus request

Continued on next page...

Table 10: (continued)

Signal Name	Direction	Description
d_hlock	output	Data bus locked transfer
d_hprot[3:0]	output	Data bus protection control
d_hsize[2:0]	output	Data bus transfer size
d_htrans[1:0]	output	Data bus transfer type
d_hwdata[63:0]	output	Data bus write data bus
d_hwrite	output	Data bus transfer direction

3.7 AXI Interface Signals

AXI interface signals provide connectivity to the AXI system bus. These signals are used only when configuration option **Bus Interface — Bus Type** is "axi" and **Bus Interface — BIU Two-port structure** is "no" . Otherwise, they should be left unconnected. They are sampled/driven in the `core_clk` clock domain when `bus_clk_en` is HIGH. Please see Section 4.2 if the bus clock frequency needs to be lower than the `core_clk` frequency.

Table 11: AXI Interface Signals

Signal Name	Direction	Description
awid[3:0]	output	Write address ID
awaddr[BIU_ADDR_WIDTH-1:0]	output	Write address
awlen[7:0]	output	Write burst length
awsize[2:0]	output	Write burst size
awburst[1:0]	output	Write burst type
awlock	output	Write lock type
awcache[3:0]	output	Write cache type
awprot[2:0]	output	Write protection type
awvalid	output	Write address valid
awready	input	Write address ready
wdata[BIU_DATA_WIDTH-1:0]	output	Write data
wstrb[(BIU_DATA_WIDTH/8)-1:0]	output	Write strobes
wlast	output	Write last
wvalid	output	Write valid
wready	input	Write ready
bid[3:0]	input	Write response ID

Continued on next page...

Table 11: (continued)

Signal Name	Direction	Description
bresp[1:0]	input	Write response
bvalid	input	Write response valid
bready	output	Write response ready
arid[3:0]	output	Read address ID
araddr[BIU_ADDR_WIDTH-1:0]	output	Read address
arlen[7:0]	output	Read burst length
arsize[2:0]	output	Read burst size
arburst[1:0]	output	Read burst type
arlock	output	Read lock type
arcache[3:0]	output	Read cache type
arprot[2:0]	output	Read protection type
arvalid	output	Read address valid
arready	input	Read address ready
rid[3:0]	input	Read ID tag
rdata[BIU_DATA_WIDTH-1:0]	input	Read data
rresp[1:0]	input	Read response
rlast	input	Read last
rvalid	input	Read valid
rready	output	Read ready

3.8 AXIx2 Interface Signals

AXIx2 interface signals provide connectivity separately to the AXI instruction and data buses. These signals are used only when configuration option **Bus Interface — Bus Type** is "axi" and **Bus Interface — BIU Two-port Structure** is "yes". Otherwise, they should be left unconnected. They are sampled/-driven in the `core_clk` clock domain when `bus_clk_en` is HIGH. Please see Section 4.2 if the bus clock frequency needs to be lower than the `core_clk` frequency.

Table 12: AXIx2 Interface Signals

Signal Name	Direction	Description
i_awid[3:0]	output	Instruction bus write address ID
i_awaddr[BIU_ADDR_WIDTH-1:0]	output	Instruction bus write address
i_awlen[7:0]	output	Instruction bus write burst length

Continued on next page...

Table 12: (continued)

Signal Name	Direction	Description
i_awsize[2:0]	output	Instruction bus write burst size
i_awburst[1:0]	output	Instruction bus write burst type
i_awlock	output	Instruction bus write lock type
i_awcache[3:0]	output	Instruction bus write cache type
i_awprot[2:0]	output	Instruction bus write protection type
i_awvalid	output	Instruction bus write address valid
i_awready	input	Instruction bus write address ready
i_wdata[BIU_DATA_WIDTH-1:0]	output	Instruction bus write data
i_wstrb[(BIU_DATA_WIDTH/8)-1:0]	output	Instruction bus write strobes
i_wlast	output	Instruction bus write last
i_wvalid	output	Instruction bus write valid
i_wready	input	Instruction bus write ready
i_bid[3:0]	input	Instruction bus write response ID
i_bresp[1:0]	input	Instruction bus write response
i_bvalid	input	Instruction bus write response valid
i_bready	output	Instruction bus write response ready
i_arid[3:0]	output	Instruction bus read address ID
i_araddr[BIU_ADDR_WIDTH-1:0]	output	Instruction bus read address
i_arlen[7:0]	output	Instruction bus read burst length
i_arsize[2:0]	output	Instruction bus read burst size
i_arburst[1:0]	output	Instruction bus read burst type
i_arlock	output	Instruction bus read lock type
i_arcache[3:0]	output	Instruction bus read cache type
i_arprot[2:0]	output	Instruction bus read protection type
i_arvalid	output	Instruction bus read address valid
i_arready	input	Instruction bus read address ready
i_rid[3:0]	input	Instruction bus read ID tag
i_rdata[BIU_DATA_WIDTH-1:0]	input	Instruction bus read data
i_rresp[1:0]	input	Instruction bus read response
i_rlast	input	Instruction bus read last
i_rvalid	input	Instruction bus read valid
i_rready	output	Instruction bus read ready
d_awid[3:0]	output	Data bus write address ID

Continued on next page...

Table 12: (continued)

Signal Name	Direction	Description
d_awaddr[BIU_ADDR_WIDTH-1:0]	output	Data bus write address
d_awlen[7:0]	output	Data bus write burst length
d_awsize[2:0]	output	Data bus write burst size
d_awburst[1:0]	output	Data bus write burst type
d_awlock	output	Data bus write lock type
d_awcache[3:0]	output	Data bus write cache type
d_awprot[2:0]	output	Data bus write protection type
d_awvalid	output	Data bus write address valid
d_awready	input	Data bus write address ready
d_wdata[BIU_DATA_WIDTH-1:0]	output	Data bus write data
d_wstrb[(BIU_DATA_WIDTH/8)-1:0]	output	Data bus write strobes
d_wlast	output	Data bus write last
d_wvalid	output	Data bus write valid
d_wready	input	Data bus write ready
d_bid[3:0]	input	Data bus write response ID
d_bresp[1:0]	input	Data bus write response
d_bvalid	input	Data bus write response valid
d_bready	output	Data bus write response ready
d_arid[3:0]	output	Data bus read address ID
d_araddr[BIU_ADDR_WIDTH-1:0]	output	Data bus read address
d_arlen[7:0]	output	Data bus read burst length
d_arsize[2:0]	output	Data bus read burst size
d_arburst[1:0]	output	Data bus read burst type
d_arlock	output	Data bus read lock type
d_arcache[3:0]	output	Data bus read cache type
d_arprot[2:0]	output	Data bus read protection type
d_arvalid	output	Data bus read address valid
d_arready	input	Data bus read address ready
d_rid[3:0]	input	Data bus read ID tag
d_rdata[BIU_DATA_WIDTH-1:0]	input	Data bus read data
d_rresp[1:0]	input	Data bus read response
d_rlast	input	Data bus read last
d_rvalid	input	Data bus read valid

Continued on next page...

Table 12: (continued)

Signal Name	Direction	Description
d_rready	output	Data bus read ready

3.9 Instruction Local Memory Interface Signals

ILM interface signals provide connectivity to the Instruction Local Memory. Two interface types, "ram" and "ahb-lite", can be configured through the configuration option **Local Memory — Local Memory Interface**. When "ram" is selected, the SRAM-style interface is configured. Otherwise, the AHB-Lite interface will be used.

SRAM-style interface signals described in Table 13 provide connectivity to the Instruction Local Memory RAM of the processor. These signals are present on the processor interface when ILM is configured to use the "ram" type. The timing diagram is shown as Figure 7.

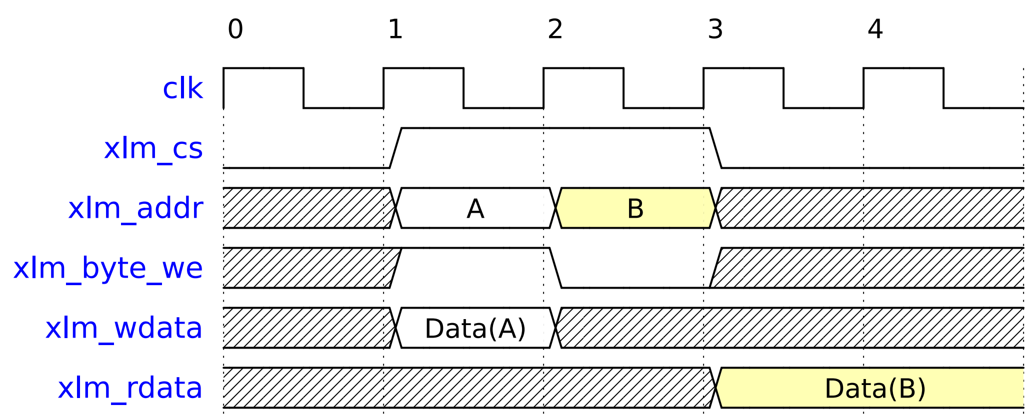


Figure 7: Timing Diagram for RAM Type LM Interface

Please see Section 22 for organization of ILM memories. See Table 14 for definitions of `ILM_RAM_AW`. See Table 15 for definitions of `ILM_RAM_DW`.

Table 13: ILM SRAM Interface Signals

Signal Name	Direction	Description
ilm_cs	output	Chip select
ilm_byte_we[7:0]	output	Byte write enable
ilm_addr[ILM_RAM_AW-1:0]	output	Address
ilm_wdata[ILM_RAM_DW-1:0]	output	Write data
ilm_rdata[ILM_RAM_DW-1:0]	input	Read data

Table 14: Instruction Local Memory Address Bit-Width

Size	ILM_RAM_AW
4KiB	9
8KiB	10
16KiB	11
32KiB	12
64KiB	13
128KiB	14
256KiB	15
512KiB	16
1MiB	17
2MiB	18
4MiB	19
8MiB	20
16MiB	21

Table 15: Instruction Local Memory Data Bit-Width

Protection Scheme	ILM_RAM_DW
none	64
parity	72
ecc	72

Table 16: ILM Byte Write Enable Mapping

BWE Bit	Protection Scheme		
	none	parity	ecc
0	wdata[7:0]	{wdata[64], wdata[7:0]}	wdata[71:0]
1	wdata[15:8]	{wdata[65], wdata[15:8]}	wdata[71:0]
2	wdata[23:16]	{wdata[66], wdata[23:16]}	wdata[71:0]
3	wdata[31:24]	{wdata[67], wdata[31:24]}	wdata[71:0]
4	wdata[39:32]	{wdata[68], wdata[39:32]}	wdata[71:0]
5	wdata[47:40]	{wdata[69], wdata[47:40]}	wdata[71:0]
6	wdata[55:48]	{wdata[70], wdata[55:48]}	wdata[71:0]
7	wdata[63:56]	{wdata[71], wdata[63:56]}	wdata[71:0]

AHB-Lite interface signals shown in Table 17 are present on the processor interface when ILM is configured to use "ahb-lite" type. Note that the AHB-Lite interface only operates in the `core_clk` clock domain.

Table 17: ILM AHB-Lite Interface Signals

Signal Name	Direction	Description
ilm_hrdata[63:0]	input	Read data bus
ilm_hready	input	Transfer done
ilm_hresp	input	Transfer response
ilm_haddr[BIU_ADDR_WIDTH-1:0]	output	Address bus
ilm_hburst[2:0]	output	Burst type
ilm_hmastlock	output	Locked transfer
ilm_hprot[3:0]	output	Protection control
ilm_hsize[2:0]	output	Transfer size
ilm_htrans[1:0]	output	Transfer type
ilm_hwdata[63:0]	output	Write data bus
ilm_hwrite	output	Transfer direction

3.10 Data Local Memory Interface Signals

DLM interface signals provide connectivity to the DATA Local Memory. Two interface types, "ram" and "ahb-lite", can be configured through the configuration option **Local Memory — Local Memory Interface**. When "ram" is selected, SRAM-style interface is configured. Otherwise, the AHB-Lite interface will be used.

SRAM-style interface signals described in Table 18 provide connectivity to the Data Local Memory RAM of the processor. These signals are present on the processor interface when DLM is configured to use the "ram" type. The timing diagram is shown as Figure 7.

Please see Section 22 for organization of DLM memories. See Table 19 for definitions of `DLM_RAM_AW`. See Table 20 for definitions of `DLM_RAM_DW`.

Table 18: Data Local Memory Interface Signals

Signal Name	Direction	Description
dlm_cs	output	Chip select
dlm_byte_we[7:0]	output	Write enable
dlm_addr[DLM_RAM_AW-1:0]	output	Address

Continued on next page...

Table 18: (continued)

Signal Name	Direction	Description
dlm_wdata[DLM_RAM_DW-1:0]	output	Write data
dlm_rdata[DLM_RAM_DW-1:0]	input	Read data

Table 19: Data Local Memory Address Bit-Width

Size	DLM_RAM_AW
4KiB	9
8KiB	10
16KiB	11
32KiB	12
64KiB	13
128KiB	14
256KiB	15
512KiB	16
1MiB	17
2MiB	18
4MiB	19
8MiB	20
16MiB	21

Table 20: Data Local Memory Data Bit-Width

Protection Scheme	DLM_RAM_DW
none	64
parity	72
ecc	72

Table 21: DLM Byte Write Enable Mapping

BWE Bit	Protection Scheme		
	none	parity	ecc
0	wdata[7:0]	{wdata[64], wdata[7:0]}	wdata[71:0]
1	wdata[15:8]	{wdata[65], wdata[15:8]}	wdata[71:0]
2	wdata[23:16]	{wdata[66], wdata[23:16]}	wdata[71:0]

Continued on next page...

Table 21: (continued)

BWE Bit	Protection Scheme		
	none	parity	ecc
3	wdata[31:24]	{wdata[67], wdata[31:24]}	wdata[71:0]
4	wdata[39:32]	{wdata[68], wdata[39:32]}	wdata[71:0]
5	wdata[47:40]	{wdata[69], wdata[47:40]}	wdata[71:0]
6	wdata[55:48]	{wdata[70], wdata[55:48]}	wdata[71:0]
7	wdata[63:56]	{wdata[71], wdata[63:56]}	wdata[71:0]

AHB-Lite interface signals shown in Table 22 are present on the processor interface when DLM is configured to use "ahb-lite" type. Note that the AHB-Lite interface only operates in the `core_clk` clock domain.

Table 22: DLM AHB-Lite Interface Signals

Signal Name	Direction	Description
dlm_hrdata[63:0]	input	Read data bus
dlm_hready	input	Transfer done
dlm_hresp	input	Transfer response
dlm_haddr[BIU_ADDR_WIDTH-1:0]	output	Address bus
dlm_hburst[2:0]	output	Burst type
dlm_hmastlock	output	Locked transfer
dlm_hprot[3:0]	output	Protection control
dlm_hsize[2:0]	output	Transfer size
dlm_htrans[1:0]	output	Transfer type
dlm_hwdata[63:0]	output	Write data bus
dlm_hwrite	output	Transfer direction

3.11 Instruction Cache Interface Signals

I-Cache interface signals provide connectivity to the I-Cache SRAMs of the processor. These signals are always present on the processor interface but they are used only when I-Cache is configured. Otherwise, they should be left unconnected. Please see Section 22 for organization of I-Cache SRAMs. See Table 24 to Table 27 for bit-width definitions.

Table 23: Instruction Cache Interface Signals

Signal Name	Direction	Description
icache_disable_init	input	Disable the initialization of I-Cache RAMs when the processor exits the reset state. Assertion of this signal is to speed up the power-gating wakeup process when the content of I-Cache SRAM is preserved during power-down.
I-Cache Tag RAM		
icache_tag*N*_cs	output	Chip select, *N*=0, 1, 2, 3
icache_tag*N*_we	output	Write enable
icache_tag*N*_addr[ICACHE_TAG_RAM_AW-1:0]	output	Address
icache_tag*N*_wdata[ICACHE_TAG_RAM_DW-1:0]	output	Write data
icache_tag*N*_rdata[ICACHE_TAG_RAM_DW-1:0]	input	Read data
I-Cache Data RAM		
icache_data*N*_cs	output	Chip select, *N*=0, 1, 2, 3
icache_data*N*_we	output	Write enable
icache_data*N*_addr[ICACHE_DATA_RAM_AW-1:0]	output	Address
icache_data*N*_wdata[ICACHE_DATA_RAM_DW-1:0]	output	Write data
icache_data*N*_rdata[ICACHE_DATA_RAM_DW-1:0]	input	Read data

Table 24: I-Cache Tag Address Bit-Width

Associativity	Size (KiB)	ICACHE_TAG_RAM_AW	
		Cache Line Size = 32	Cache Line Size = 64
1	4	7	6
1	8	8	7
1	16	9	8
1	32	10	9
1	64	11	10
2	8	7	6
2	16	8	7
2	32	9	8
2	64	10	9

Continued on next page...

Table 24: (continued)

Associativity	Size (KiB)	ICACHE_TAG_RAM_AW Cache Line Size = 32	ICACHE_TAG_RAM_AW Cache Line Size = 64
4	8	6	5
4	16	7	6
4	32	8	7
4	64	9	8

Table 25: I-Cache Tag Data Bit-Width

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	ICACHE_TAG_RAM_DW
none	1	4	N/A	BIU_ADDR_WIDTH-9
none	1	8	N/A	BIU_ADDR_WIDTH-9
none	1	16	N/A	BIU_ADDR_WIDTH-9
none	1	32	N/A	BIU_ADDR_WIDTH-9
none	1	64	N/A	BIU_ADDR_WIDTH-9
none	2	8	N/A	BIU_ADDR_WIDTH-9
none	2	16	N/A	BIU_ADDR_WIDTH-9
none	2	32	N/A	BIU_ADDR_WIDTH-9
none	2	64	N/A	BIU_ADDR_WIDTH-9
none	4	8	N/A	BIU_ADDR_WIDTH-8
none	4	16	N/A	BIU_ADDR_WIDTH-9
none	4	32	N/A	BIU_ADDR_WIDTH-9
none	4	64	N/A	BIU_ADDR_WIDTH-9
parity	1	4	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	4	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	1	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	1	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	1	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	1	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1

Continued on next page. . .

Table 25: (continued)

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	ICACHE_TAG_RAM_DW
parity	2	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	2	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	2	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	2	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	2	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	2	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	2	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	2	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	4	8	(BIU_ADDR_WIDTH-8) $\leq$ 32	BIU_ADDR_WIDTH-4
parity	4	8	(BIU_ADDR_WIDTH-8) > 32	BIU_ADDR_WIDTH
parity	4	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	4	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	4	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	4	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	4	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	4	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	1	4	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	4	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	1	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	1	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	1	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	1	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	2	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	2	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	2	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	2	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	2	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	2	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1

Continued on next page...

Table 25: (continued)

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	ICACHE_TAG_RAM_DW
ecc	2	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	2	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	4	8	(BIU_ADDR_WIDTH-8) $\leq$ 32	BIU_ADDR_WIDTH-1
ecc	4	8	(BIU_ADDR_WIDTH-8) > 32	BIU_ADDR_WIDTH
ecc	4	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	4	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	4	32	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	4	32	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	4	64	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	4	64	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1

Table 26: I-Cache Data Address Bit-Width

Associativity	Size (KiB)	ICACHE_DATA_RAM_AW
1	4	10
1	8	11
1	16	12
1	32	13
1	64	14
2	8	10
2	16	11
2	32	12
2	64	13
4	8	9
4	16	10
4	32	11
4	64	12

Table 27: I-Cache Data Bit-Width

Protection Scheme	ICACHE_DATA_RAM_DW
none	32
parity	36

Continued on next page...

Table 27: (continued)

Protection Scheme	ICACHE_DATA_RAM_DW
ecc	39

3.12 Data Cache Interface Signals

D-Cache interface signals provide connectivity to D-Cache SRAMs of the processor. These signals are always present on the processor interface but they are used only when D-Cache is configured. Otherwise, they should be left unconnected. Please see Section 22 for organization of D-Cache SRAMs. See Table 29 to Table 33 for bit-width definitions.

For configurations with Parity/ECC support, how the byte-write-enables control the corresponding data bits are described in Table 33. The data bits will need to be regrouped accordingly before connecting to the SRAM data bus for proper parity/ECC operations unless the byte-write-enables are implemented using bit-writes.

Table 28: Data Cache Interface Signals

Signal Name	Direction	Description
dcache_disable_init	input	Disable the initialization of D-Cache RAMs when the processor exits the reset state. Assertion of this signal is to speed up the power-gating wakeup process when the content of D-Cache SRAM is preserved during power-down.
D-Cache Tag RAM		
dcache_tag*N*_cs	output	Chip select, *N*=0, 1, 2, 3
dcache_tag*N*_bit_we[DCACHE_TAG_RAM_DW-1:0]	output	Bit write enable
dcache_tag*N*_addr[DCACHE_TAG_RAM_AW-1:0]	output	Address
dcache_tag*N*_wdata[DCACHE_TAG_RAM_DW-1:0]	output	Write data
dcache_tag*N*_rdata[DCACHE_TAG_RAM_DW-1:0]	input	Read data
D-Cache Data RAM		
dcache_data*N*_cs[0:0]	output	
dcache_data*N*_we[DCACHE_DATA_RAM_BWEW-1:0]	output	Byte write enable
dcache_data*N*_addr[DCACHE_DATA_RAM_AW-1:0]	output	Address

Continued on next page. . .

Table 28: (continued)

Signal Name	Direction	Description
dcache_data*N*_wdata[DCACHE_DATA_RAM_DW-1:0]	output	Write data
dcache_data*N*_rdata[DCACHE_DATA_RAM_DW-1:0]	input	Read data

Table 29: D-Cache Tag Address Bit-Width

Associativity	Size (KiB)	DCACHE_TAG_RAM_AW	
		Cache Line Size = 32	Cache Line Size = 64
1	4	7	6
1	8	8	7
1	16	9	8
1	32	10	9
1	64	11	10
2	8	7	6
2	16	8	7
2	32	9	8
2	64	10	9
4	8	6	5
4	16	7	6
4	32	8	7
4	64	9	8

Table 30: D-Cache Tag Data Bit-Width

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	DCACHE_TAG_RAM_DW
none	1	4	N/A	BIU_ADDR_WIDTH-9
none	1	8	N/A	BIU_ADDR_WIDTH-10
none	1	16	N/A	BIU_ADDR_WIDTH-11
none	1	32	N/A	BIU_ADDR_WIDTH-12
none	1	64	N/A	BIU_ADDR_WIDTH-13
none	2	8	N/A	BIU_ADDR_WIDTH-9
none	2	16	N/A	BIU_ADDR_WIDTH-10
none	2	32	N/A	BIU_ADDR_WIDTH-11
none	2	64	N/A	BIU_ADDR_WIDTH-12

Continued on next page...

Table 30: (continued)

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	DCACHE_TAG_RAM_DW
none	4	8	N/A	BIU_ADDR_WIDTH-8
none	4	16	N/A	BIU_ADDR_WIDTH-9
none	4	32	N/A	BIU_ADDR_WIDTH-10
none	4	64	N/A	BIU_ADDR_WIDTH-11
parity	1	4	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	1	4	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	1	8	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-6
parity	1	8	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
parity	1	16	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-7
parity	1	16	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3
parity	1	32	(BIU_ADDR_WIDTH-12) $\leq$ 32	BIU_ADDR_WIDTH-8
parity	1	32	(BIU_ADDR_WIDTH-12) > 32	BIU_ADDR_WIDTH-4
parity	1	64	(BIU_ADDR_WIDTH-13) $\leq$ 32	BIU_ADDR_WIDTH-9
parity	1	64	(BIU_ADDR_WIDTH-13) > 32	BIU_ADDR_WIDTH-5
parity	2	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	2	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	2	16	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-6
parity	2	16	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
parity	2	32	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-7
parity	2	32	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3
parity	2	64	(BIU_ADDR_WIDTH-12) $\leq$ 32	BIU_ADDR_WIDTH-8
parity	2	64	(BIU_ADDR_WIDTH-12) > 32	BIU_ADDR_WIDTH-4
parity	4	8	(BIU_ADDR_WIDTH-8) $\leq$ 32	BIU_ADDR_WIDTH-4
parity	4	8	(BIU_ADDR_WIDTH-8) > 32	BIU_ADDR_WIDTH
parity	4	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-5
parity	4	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
parity	4	32	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-6
parity	4	32	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
parity	4	64	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-7
parity	4	64	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3
ecc	1	4	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	1	4	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1

Continued on next page...

Table 30: (continued)

Protection Scheme	Associativity	Size (KiB)	Protection Width > 32	DCACHE_TAG_RAM_DW
ecc	1	8	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-3
ecc	1	8	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
ecc	1	16	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-4
ecc	1	16	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3
ecc	1	32	(BIU_ADDR_WIDTH-12) $\leq$ 32	BIU_ADDR_WIDTH-5
ecc	1	32	(BIU_ADDR_WIDTH-12) > 32	BIU_ADDR_WIDTH-4
ecc	1	64	(BIU_ADDR_WIDTH-13) $\leq$ 32	BIU_ADDR_WIDTH-6
ecc	1	64	(BIU_ADDR_WIDTH-13) > 32	BIU_ADDR_WIDTH-5
ecc	2	8	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	2	8	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	2	16	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-3
ecc	2	16	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
ecc	2	32	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-4
ecc	2	32	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3
ecc	2	64	(BIU_ADDR_WIDTH-12) $\leq$ 32	BIU_ADDR_WIDTH-5
ecc	2	64	(BIU_ADDR_WIDTH-12) > 32	BIU_ADDR_WIDTH-4
ecc	4	8	(BIU_ADDR_WIDTH-8) $\leq$ 32	BIU_ADDR_WIDTH-1
ecc	4	8	(BIU_ADDR_WIDTH-8) > 32	BIU_ADDR_WIDTH
ecc	4	16	(BIU_ADDR_WIDTH-9) $\leq$ 32	BIU_ADDR_WIDTH-2
ecc	4	16	(BIU_ADDR_WIDTH-9) > 32	BIU_ADDR_WIDTH-1
ecc	4	32	(BIU_ADDR_WIDTH-10) $\leq$ 32	BIU_ADDR_WIDTH-3
ecc	4	32	(BIU_ADDR_WIDTH-10) > 32	BIU_ADDR_WIDTH-2
ecc	4	64	(BIU_ADDR_WIDTH-11) $\leq$ 32	BIU_ADDR_WIDTH-4
ecc	4	64	(BIU_ADDR_WIDTH-11) > 32	BIU_ADDR_WIDTH-3

Table 31: D-Cache Data Address Bit-Width

Associativity	Size (KiB)	DCACHE_DATA_RAM_AW
1	4	9
1	8	10
1	16	11
1	32	12
1	64	13

Continued on next page. . .

Table 31: (continued)

Associativity	Size (KiB)	DCACHE_DATA_RAM_AW
2	8	9
2	16	10
2	32	11
2	64	12
4	8	8
4	16	9
4	32	10
4	64	11

Table 32: D-Cache Data Bit-Width

Protection Scheme	DCACHE_DATA_RAM_DW	DCACHE_DATA_RAM_BWEW
none	64	8
parity	72	8
ecc	72	8

Table 33: D-Cache Byte Write Enable Mapping

BWE Bit	Protection Scheme		
	none	parity	ecc
0	wdata[7:0]	{wdata[64], wdata[7:0]}	wdata[71:0]
1	wdata[15:8]	{wdata[65], wdata[15:8]}	wdata[71:0]
2	wdata[23:16]	{wdata[66], wdata[23:16]}	wdata[71:0]
3	wdata[31:24]	{wdata[67], wdata[31:24]}	wdata[71:0]
4	wdata[39:32]	{wdata[68], wdata[39:32]}	wdata[71:0]
5	wdata[47:40]	{wdata[69], wdata[47:40]}	wdata[71:0]
6	wdata[55:48]	{wdata[70], wdata[55:48]}	wdata[71:0]
7	wdata[63:56]	{wdata[71], wdata[63:56]}	wdata[71:0]

3.13 Local Memory Slave Port Signals

Slave Port signals allow external agents to access the local memories of the processor through the AHB interface. These signals are always present on the processor interface but they are used only when configuration option **Local Memory — Slave Port Support** is "yes". Otherwise, they should be left unconnected. They are sampled/driven in the `core_clk` clock domain when `slv_clk_en` is HIGH. Please see Section 4.2 if the Slave Port clock frequency needs to be lower than the `core_clk` frequency.

Please see Section 2.7.4 for the value of `SLAVE_PORT_DATA_WIDTH` which is used below.

Table 34: AHB Local Memory Slave Port Signals

Signal Name	Direction	Description
slv_clk_en	input	This is a `core_clk` clock domain signal indicating that the data from the AHB Slave Port clock domain can be sampled at the coming rising edge of `core_clk`. See Section 4.2 for details.
slv_hsel	input	Bus grant
slv_huser	input	Select ILM/DLM (0:ILM, 1:DLM)
slv_hready	input	Ready in
slv_hreadyout	output	Ready out
slv_haddr[31:0]	input	Address bus
slv_hburst[2:0]	input	Burst type
slv_hprot[3:0]	input	Protection control
slv_hsize[2:0]	input	Transfer size
slv_htrans[1:0]	input	Transfer type
slv_hwrite	input	Transfer direction
slv_hwdata[SLAVE_PORT_DATA_WIDTH-1:0]	input	Write data bus
slv_hrdata[SLAVE_PORT_DATA_WIDTH-1:0]	output	Read data bus
slv_hresp[1:0]	output	Transfer response

3.14 BTB Interface Signals

BTB interface signals provide connectivity to the BTB SRAMs of the processor. These signals are always present on the process interface but they are used only when BTB is configured. Otherwise, they should be left unconnected. BTB in NX25(F) is 2-way associative. Please see Section 22 for organization of BTB SRAMs. See Table 36 for definition of `BTB_RAM_ADDR_WIDTH`.

Table 35: BTB Memory Interface Signals

Signal Name	Direction	Description
btb0_cs	output	Chip select for BTB memory 0
btb0_we	output	Write enable for BTB memory 0
btb0_addr[BTB_RAM_ADDR_WIDTH-1:0]	output	Address for BTB memory 0
btb0_wdata[41:0]	output	Write data for BTB memory 0
btb0_rdata[41:0]	input	Read data for BTB memory 0
btb1_cs	output	Chip select for BTB memory 1
btb1_we	output	Write enable for BTB memory 1
btb1_addr[BTB_RAM_ADDR_WIDTH-1:0]	output	Address for BTB memory 1
btb1_wdata[41:0]	output	Write data for BTB memory 1
btb1_rdata[41:0]	input	Read data for BTB memory 1

Table 36: BTB RAM Address Bit-Width

BTB Size	BTB_RAM_ADDR_WIDTH	RAM Dimension
32	4	16×42
64	5	32×42
128	6	64×42
256	7	128×42

3.15 ACE Signals

ACE signals are a set of signals for interfaces required by ACE custom instructions. Specifically, new interface signals will appear on the port list of NX25(F) when ACE custom memories (ACM) with AHB-/AXI-type and ACE custom ports (ACP) are used. The interface signals for AHB-ACM, AXI-ACM and ACP are listed in Table 37 , Table 38 and Table 39, respectively.

Table 37: ACE AHB-ACM Interface Signals

Signal Name	Direction	Description
ace_ahb*M*_hgrant	input	AHB port *M* bus grant.
ace_ahb*M*_hrdata[X-1:0]	input	AHB port *M* read data bus, where X is the specified data width.
ace_ahb*M*_hready	input	AHB port *M* transfer done.
ace_ahb*M*_hresp[1:0]	input	AHB port *M* transfer response.
ace_ahb*M*_haddr[Y-1:0]	output	AHB port *M* address bus, where Y is the BIU address width of NX25(F).
ace_ahb*M*_hburst[2:0]	output	AHB port *M* burst type. The burst type generated by AHB-ACMs will always be SINGLE (0).
ace_ahb*M*_hbusreq	output	AHB port *M* bus request.
ace_ahb*M*_hlock	output	AHB port *M* lock transfer. AHB-ACMs will not generate lock transfers so the value of this signal is a constant zero.
ace_ahb*M*_hprot[3:0]	output	AHB port *M* protection control. All AHB-ACM transfers are defined to be privileged data accesses, so the value of this signal is a constant 3 ('b0011).
ace_ahb*M*_hsize[2:0]	output	AHB port *M* transfer size.
ace_ahb*M*_htrans[1:0]	output	AHB port *M* transfer type.
ace_ahb*M*_hwdata[X-1:0]	output	AHB port *M* write data bus.
ace_ahb*M*_hwrite	output	AHB port *M* transfer direction.

Table 38: ACE AXI-ACM Interface Signals

Signal Name	Direction	Description
ace_axi*M*_awid[3:0]	output	AXI port M write address ID.
ace_axi*M*_awaddr[Y-1:0]	output	AXI port M write address, where Y is the BIU address width of the baseline processor.
ace_axi*M*_awlen[7:0]	output	AXI port M write burst length. The value will always be 1 for transferring one data at a time.
ace_axi*M*_awsize[2:0]	output	AXI port M write burst size.
ace_axi*M*_awburst[1:0]	output	AXI port M write burst type. The burst type will always be INCR ('b01).
ace_axi*M*_awlock	output	AXI port M write lock type. The atomic accessing type will always be normal access (0).

Continued on next page. . .

Table 38: (continued)

Signal Name	Direction	Description
ace_axi*M*_awcache[3:0]	output	AXI port M write cache type. All transactions do not need to be looked up in a cache and must not be modified ('b0000).
ace_axi*M*_awport[2:0]	output	AXI port M write protection type. The access permission will always be data access, non-secure access and privileged access ('b011).
ace_axi*M*_awvalid	output	AXI port M write address valid.
ace_axi*M*_awready	input	AXI port M write address ready.
ace_axi*M*_wdata[X-1:0]	output	AXI port M write data, where X is the specified data width.
ace_axi*M*_wstrb[(X/8)-1:0]	output	AXI port M write strobes, where X is the specified data width.
ace_axi*M*_wlast	output	AXI port M write last.
ace_axi*M*_wvalid	output	AXI port M write valid.
ace_axi*M*_wready	input	AXI port M write ready.
ace_axi*M*_bid[3:0]	input	AXI port M write response ID.
ace_axi*M*_bresp[1:0]	input	AXI port M write response.
ace_axi*M*_bvalid	input	AXI port M write response valid.
ace_axi*M*_bready	output	AXI port M write response ready.
ace_axi*M*_arid[3:0]	output	AXI port M read address ID.
ace_axi*M*_araddr[Y-1:0]	output	AXI port M read address, where Y is the BIU address width of the baseline processor.
ace_axi*M*_arlen[7:0]	output	AXI port M read burst length. The value will always be 1 for transferring one data at a time.
ace_axi*M*_arsize[2:0]	output	AXI port M read burst size.
ace_axi*M*_arburst[1:0]	output	AXI port M read burst type. The burst type will always be INCR ('b01)
ace_axi*M*_arlock	output	AXI port M read lock type. The atomic accessing type will always be normal access (0).
ace_axi*M*_arcache[3:0]	output	AXI port M read cache type. All transactions do not need to be looked up in a cache and must not be modified ('b0000).

Continued on next page...

Table 38: (continued)

Signal Name	Direction	Description
ace_axi*M*_arprot[2:0]	output	AXI port M read protection type. The access permission will always be data access, non-secure access and privileged access ('b011).
ace_axi*M*_arvalid	output	AXI port M read address valid.
ace_axi*M*_arready	input	AXI port M read address ready.
ace_axi*M*_rid[3:0]	input	AXI port M read ID tag.
ace_axi*M*_rdata[X-1:0]	input	AXI port M read data, where X is the specified data width.
ace_axi*M*_rresp[1:0]	input	AXI port M read response.
ace_axi*M*_rlast	input	AXI port M read last.
ace_axi*M*_rvalid	input	AXI port M read valid.
ace_axi*M*_rready	output	AXI port M read ready.

Table 39: ACE ACP Interface Signals

Signal Name	Direction	Description
ace_port_*NAME*_*N*[X-1:0]	input or output	ACP*NAME* (port *N*), where X is the specified data width. (N is necessary when ACP number > 1)

4 Reset and Clocking Scheme

4.1 Reset

NX25(F) uses input signal `core_reset_n` to reset the corresponding core clock domain. The reset signal(s) should be synchronized to the `core_clk` clock domain before connecting to NX25(F).

Regarding the *Andes Custom Extension* feature, the ACE Engine is also reset by `core_reset_n` since it also operates in the `core_clk` clock domain.

In addition, to maintain proper reset ordering, `core_reset_n` should only be released after the release of the reset signal to the bus clock domain, even though NX25(F) does not take the reset signal to the bus clock domain as its input. Figure 8 illustrates a reference design for reset synchronization.

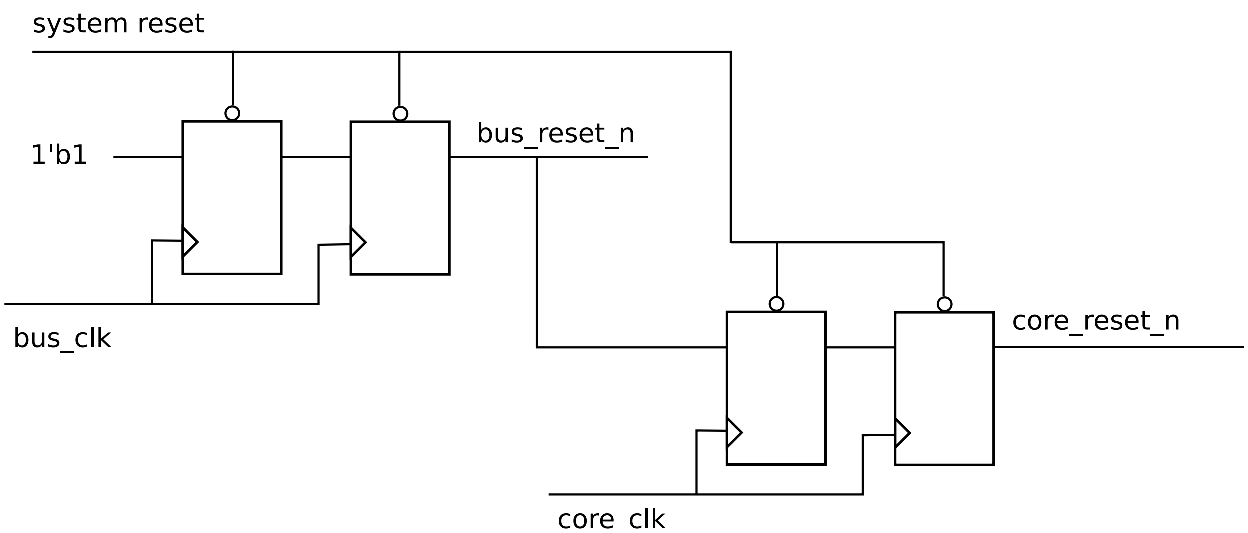


Figure 8: Reference Design for Reset Synchronization

4.2 Clock Domains

NX25(F) provides only one clock domain: `core_clk`. However, its bus interface signals may be operating in a separate synchronous bus clock domain. The synchronous bus clock is a virtual clock to the NX25(F) design. Input signal `bus_clk_en` serves as the clock enable signal for generating the virtual clock. `bus_clk_en` is a `core_clk` domain signal and should be asserted for one `core_clk` cycle before the rising edge of the bus clock, as shown in Figure 9. NX25(F) uses `bus_clk_en` to determine valid cycles to sample/drive the bus interface signals.

The Slave Port may also be operating in separate synchronous clock domains. The synchronous Slave Port clock is a virtual clock to the NX25(F) design, and input signal `slv_clk_en` serves as the clock enable signal for generating the virtual clock. Similarly, `slv_clk_en` should be asserted for one `core_clk` cycle before the rising edge of the virtual Slave Port clock.

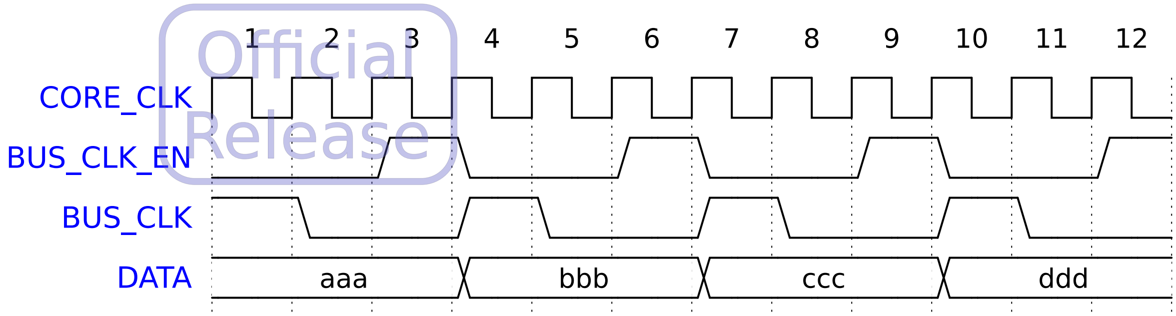


Figure 9: BUS_CLK_EN Waveform for N:1 (3:1) Clock Ratio

Detailed clock domain constraints can be found in the synthesis script `timing_con.tcl` as described in Section 24.3.

4.3 Race-Free Clock and Reset Generation Considerations

NX25(F) applies two clock domains, the CORE_CLK domain and the bus clock domain. The RTL modeling of flops assumes zero clock-to-Q delay (no #-delay in the nonblocking assignments). Because of no #-delay flop modeling, special care must be taken to generate clocks so that there are no clock skew problems for signals that cross the two clock domains. Otherwise, simulator event ordering may cause races in simulation that are similar to hold time violations.

Either one of the rules below should be followed to avoid simulator event ordering problems crossing clock domains:

1. All clocks are generated by blocking assignments in the same initial block.
2. When generating the bus clock by dividing CORE_CLK through flops, CORE_CLK should also be delayed through a non-blocking assignment to better align the two clock edges. See Figure 10 below for detailed information.

These rules are also applicable to reset signals similarly.

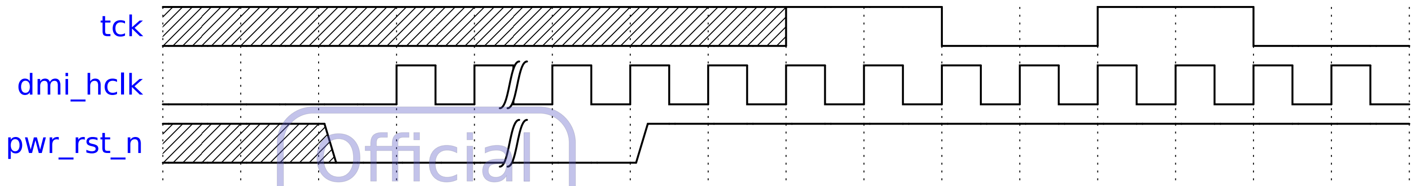
```
// core_clk and bus_clk clock ratio is 2:1
reg     root_clk;
reg     root_rstn;
reg     core_clk;
reg     core_rstn;
reg     bus_clk;
reg     bus_clk_en;

initial begin
        root_clk = 1'b0;
        reset_n = 1'b0;
        #(PERIOD/2) root_clk = 1'b1; #(PERIOD/2) root_clk = 1'b0;
        #(PERIOD/2) root_clk = 1'b1; #(PERIOD/2) root_clk = 1'b0;
        rstn = 1'b1;
        forever #(PERIOD/2) root_clk = ~root_clk;
end


// ----------------------------------------------------------------
// use nonblocking assignment to align core_clk edge with bus_clk edg
e
// ----------------------------------------------------------------
always @(root_clk) begin
        core_clk <= root_clk;
end

always @(root_rstn) begin
        core_rstn <= root_rstn;
end


// divide clk by 2
always @(posedge root_clk or negedge root_rstn) begin
        if (!root_rstn)
                bus_clk <= 1'b1;
        else
                bus_clk <= ~bus_clk;
end

always @(posedge core_clk or negedge core_rstn) begin
        if (!core_rstn)
                bus_clk_en <= 1'b0;
        else
                bus_clk_en <= ~bus_clk_en;
```

Figure 10: Race-Free CORE_CLK/HCLK Generation

4.4 Clock and Reset Relationships

Some design blocks in the AE350 platform need special considerations for the associated clocks and resets. In this section, waveforms and related descriptions for relationships of clocks and resets are provided to ease the integration work.

4.4.1 NCEJDTM200

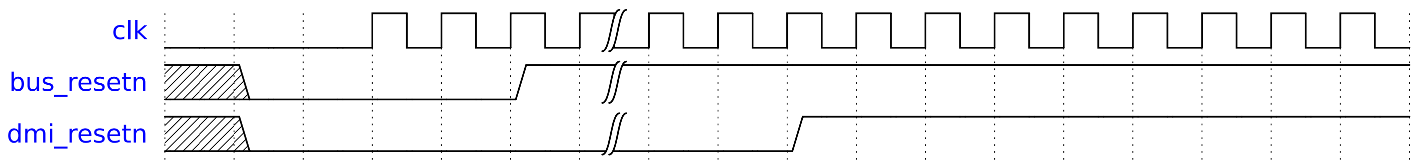


Note

1. Clock `tck` is “Don’t Care” when `pwr_rst_n` is active.

4.4.2 NCEPLDM200

4.4.2.1 Power-on Phase

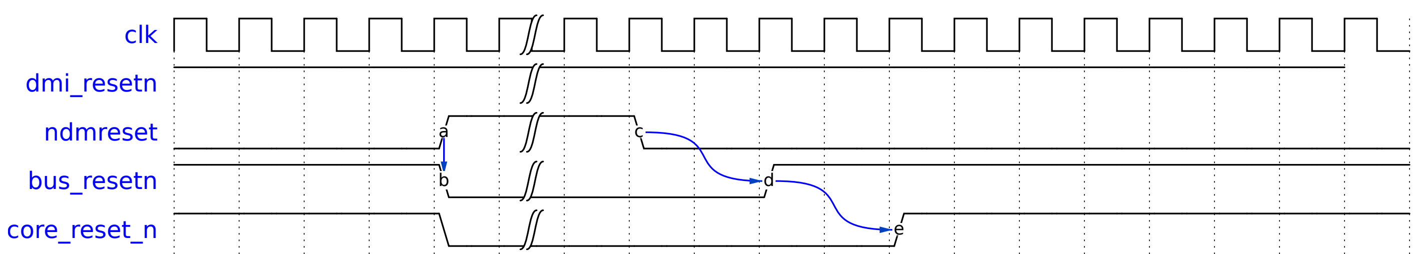


In the current implementation, `clk` is connected to the bus clock, which `bus_resetn` is synchronized to.

`dmi_resetn` comes from NCEJDTM200. It is asserted by the power-on reset. In addition, the external debugger can also write NCEJDTM200 registers to assert `dmi_resetn`. In the power-on phase, `dmi_resetn` may be active much longer than `bus_resetn` since the control for `dmi_resetn` is operating in the TCK domain and the control signal needs to be synchronized over to the `dmi_hclk` domain. But the order between asserting `bus_resetn` and asserting `dmi_resetn` is not important.

Currently, `dmi_hclk` of NCEJDTM200 is also connected to the bus clock.

4.4.2.2 External Debugger Triggered Reset



a. The external debugger writes the `dmcontrol` register of NCEPLDM200 to assert the `ndmreset` signal to reset everything in the debug target except the debug module.

b. The `ndmreset` signal is propagated to the system reset control circuit, and activates the system bus reset (`bus_resetn`) and processor reset (`core_reset_n`) immediately.

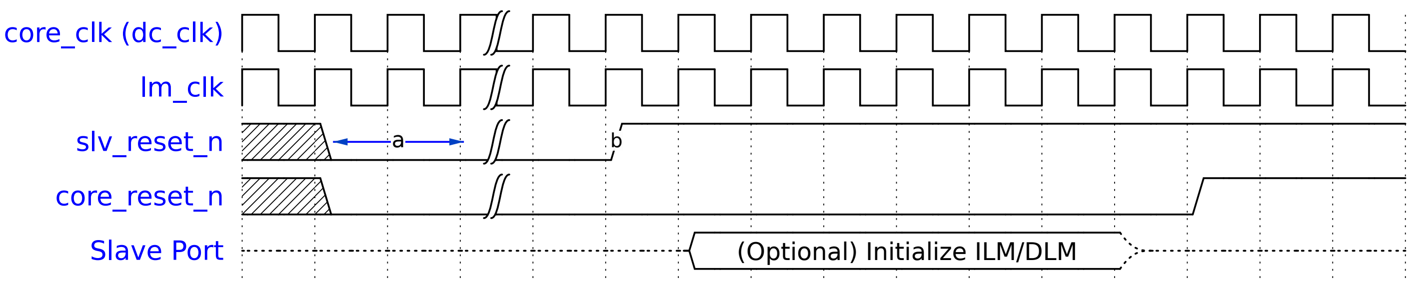
c. The external debugger programs `dmcontrol` again to deassert `ndmreset`.

d. Some cycles later, `bus_resetn` is released after the release of `ndmreset`.

e. Some cycles later, `core_reset_n` is released after the release of `bus_resetn`.

4.4.3 nx25_core_top

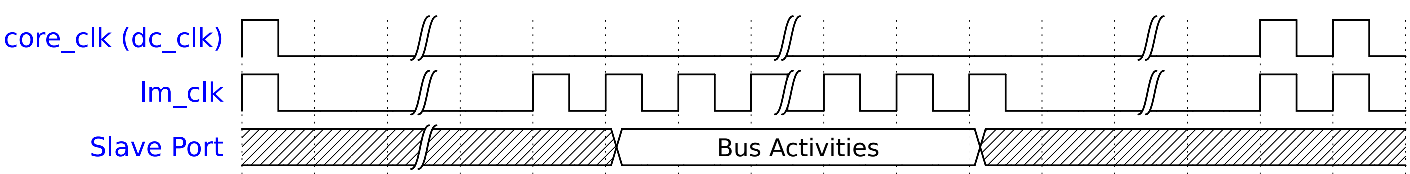
4.4.3.1 Power-on Phase



a. `slv_reset_n` and `core_reset_n` should be active simultaneously for some period of time to properly reset the control circuits of local memories.

b. If boot-from-ILM is needed, `slv_reset_n` should be deasserted before `core_reset_n` is deasserted, so that ILM/DLM can be filled with valid data through the slave port before the processor boots up.

4.4.3.2 Access LM While the Processor Is Inactive



If `lm_clk` is active, the local memories can still be accessed through the slave port while the processor is idle or under reset.

5 Instruction Set Overview

5.1 Introduction

The processor implements the *RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2* and the *RISC-V Bit-Manipulation ISA-extensions (TD013) V1.0.0*. The following instruction sets are implemented:

- RV64I base integer instruction set
- RISC-V "A" standard extension
- RISC-V "B" standard extension
- RISC-V "C" standard extension
- RISC-V "M" standard extension
- AndeStar V5 instruction extension

For detailed information, please see the *RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2*, the *RISC-V Bit-Manipulation ISA-extensions (TD013) V1.0.0* and the *AndeStar V5 Instruction Extension Specification (UM165)*.

5.2 Integer Registers

Table 40 lists all general-purpose integer registers.

Table 40: Integer Registers

Register	Signal Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6–x7	t1–t2	Temporaries
x8	s0/fp	Saved register/frame pointer
x9	s1	Saved register
x10–x11	a0–a1	Function arguments/return values

Continued on next page...

Table 40: (continued)

Register	Signal Name	Description
x12–x17	a2–a7	Function arguments
x18–x27	s2–s11	Saved registers
x28–x31	t3–t6	Temporaries

5.3 Atomic Instructions

The RVA extension includes load-reserved/store-conditional and atomic memory operation (AMO) instructions.

5.3.1 Load-Reserved/Store-Conditional Instruction

The processor tracks at most one physical address location for `LR`-`SC` instructions at a time. The reservation made by the `LR` instruction is canceled after any memory operation or exception happens. The address of `SC` instructions must match the reserved address for `SC` to succeed.

5.3.2 Atomic Memory Operation Instruction

An atomic memory operation is expanded to `LR`-modify-`SC` sequences in the processor. The memory content is first loaded with the `LR` instruction, then the required operation is performed on the retrieved data, and the final result is written back to the memory by the `SC` instruction. If the `SC` instruction fails, the sequence will be retried until it succeeds.

5.4 Misaligned Memory Access

The processor implements the misaligned memory access to support accessing misaligned addresses without triggering any Address Misaligned exceptions.

By controlling the `mmisc_ctl` CSR, the scheme can be enabled or disabled. Please see Section 15.10.7 for details.

5.4.1 Exceptions

When the misaligned memory access scheme is enabled, Access Fault exceptions will still be triggered under the following cases:

- Accesses to device regions
- Accesses across two different memory regions
- Atomic accesses

If the misaligned memory access scheme is disabled, misaligned accesses will trigger Access Fault exceptions or Address Misaligned exceptions. Access fault exceptions are triggered when the following cases occur:

- Atomic accesses
- Address is located in a device region

Other misaligned accesses trigger Address Misaligned exceptions.

5.5 Floating-Point ISA Extension

The processor supports the "F" and "D" Standard Extensions for accelerating the performance of floating-point heavy applications. The supported configuration is indicated in the `misa` (Machine ISA) configuration register.

The supported FPU features include:

- Fully pipelined MAC instructions
- Hardware subnormal handling
- All rounding modes

6 Physical Memory Attributes

6.1 Introduction

Memory locations can have various attributes associated with them. Any location is basically categorized into either one of the two types: device region and (normal) memory region. While memory regions are cacheable locations, device regions are non-cacheable locations where accesses to these locations may cause side effects.

Memory locations that are not defined to be device regions are (cacheable) memory regions.

For the write to cacheable memory regions, write-back and write-through policies are supported. Therefore, cacheable memory regions can be further divided into write-back regions and write-through regions. Write-back regions are memory locations where the memory write only updates the D-Cache entry initially. The next-level memory will only be updated when the corresponding D-Cache entry is about to be overwritten by another block of data. Write-through regions are memory locations where the memory write updates both the D-Cache entry and the next-level memory.

The write-miss policies (write-allocate and write-no-allocate) decide whether to allocate the D-Cache entry on write misses. The write-miss policy for write-back regions is write-allocate and the write-miss policy for write-through regions is write-no-allocate.

Table 41 summarizes the write behavior of these two regions.

Table 41: Write Behavior in Cacheable Regions

Write Policy	Write-Hit	Write-Miss
Write-Back	Write to D-Cache	Write-allocate
Write-Through	Write-through	Write-no-allocate

Write-through regions and ILM/DLM/DEVICE regions should not overlap with each other. The behavior is UNDEFINED when these regions overlap.

NX25(F) provides the static setting for physical memory attributes through the Device Region and Write-through Region configuration options. If a physical address matches neither one of the regions, this address will be treated cacheable and the write-back policy will be used.

6.2 Memory Access Ordering

Accesses to device regions are strongly-ordered. They are guaranteed to be non-speculative and issued in program order. An access to a device region is not issued until all preceding accesses to device regions are finished.

Note

Loads to device regions are blocking and the processor pipeline pauses until data returns. There can be only one single bus write transaction outstanding for stores to device regions but these stores do not necessarily block the processor pipeline. Store data retire to the store buffer first and get committed to memory when the corresponding entry is the oldest one in the store buffer. As long as ordering rules are not violated, instructions after device stores may proceed without being blocked by the outstanding bus write transaction. Subsequent device load/stores will block the pipeline and wait until the store buffer is empty. Furthermore, stores will also block the pipeline when the store buffer is full. (The size of the store buffer is 4-entries). A `FENCE` instruction can be used to explicitly wait for the outstanding device store to finish.

In particular, when an device store is outstanding, both subsequent uncached and cacheable stores may proceed by retiring data to store buffer; cacheable loads are also not blocked by the outstanding device store as well. Uncached stores retire data into the store buffer and the actual write request is sent out to the bus only when the corresponding entries become the oldest one in the store buffer. Cacheable stores may retire data into the store buffer if either they hit D-Cache, or they cause the first D-Cache miss, or when D-Cache is off or not configured. Similarly, cacheable loads may proceed if they hit D-Cache, or they cause the first D-Cache miss, or when they hit the store buffer.

On the other hand, accesses to the memory regions could be speculative and the order of accessing memory regions is not guaranteed. A load access to a cacheable memory region might bypass an earlier store access if there is no data dependency. The store could be either to a memory region or even to a device region. In such a scenario, explicit `FENCE` instructions are required to guarantee the order.

Table 42 shows ordering of two instructions A and B, where $A < B$ (A comes earlier than B) in program order.

Table 42: Memory Access Ordering

A<B in Program Order	B	
A	Normal Memory	Device
Normal Memory	-	-
Device	-	<

7 Local Memory

7.1 Introduction

Local memories store data or instructions that might either be accessed frequently or require deterministic access latency, such as interrupt service routines, system calls, video data, real-time systems, etc. Local memories are *memories* and accesses to them are treated the same as to the cacheable memory space. It is not suitable to map device registers in the local memories.

The processor supports both instruction local memory (ILM) and data local memory (DLM). They are dedicated address spaces that are independent of the memory subsystem. Accesses to them bypass the cache and memory subsystems to achieve minimal latency. The Local Memory Base Address is specified by processor configuration options described in Section 2. The details of local memory usages are described in the subsequent sections.

7.2 Local Memory Spaces

The processor supports three address spaces: the instruction local memory, the data local memory and the system bus (AHB/AXI) address spaces. The ILM address space is defined by **ILM Size** and **ILM Base** configuration options, and the DLM address space is defined by **DLM Size** and **DLM Base** configuration options. The base address of the Andes local memory should be aligned to its size (a power-of-2 size). See Section 2 for more information regarding the configuration parameters. Any addresses outside the local memory address spaces belong to the system bus address space.

Instruction fetches go to the instruction local memory or the system bus while load/store data accesses access all three regions of spaces. The address spaces for ILM and DLM should not overlap with each other to achieve maximum compatibility across Andes processor products. The exact address space access priorities for the processor are defined in Table 43 for instruction fetches and Table 44 for load/store data accesses.

It is not recommended to set the instruction local memory and the data local memory to have the same base address. Otherwise, UNPREDICTABLE behavior might happen.

Table 43: Priorities for Instruction Fetches

Address Hit the ILM Space	Address Hit the DLM Space	Actual Space Accessed
No	No	AHB/AXI address space
No	Yes	AHB/AXI address space

Continued on next page...

Table 43: (continued)

Address Hit the ILM Space	Address Hit the DLM Space	Actual Space Accessed
Yes	No	ILM
Yes	Yes	ILM (not recommended; the ILM and DLM spaces should not overlap)

Table 44: Priorities for Data Accesses

Address Hit the ILM Space	Address Hit the DLM Space	Actual Space Accessed
No	No	AHB/AXI address space
No	Yes	DLM
Yes	No	ILM
Yes	Yes	DLM (not recommended; the ILM and DLM spaces should not overlap)

7.3 Local Memory Address Range

The local memory address ranges are listed in Table 45. LM_BASE represents the base address field of the ILM and DLM local memory base address system registers (`milmb.IBPA` and `mdlmb.DBPA`).

Table 45: Local Memory Address Range (for ILM and DLM)

LM Size	Start	End
4KiB	(LM_BASE[63:12]<<12)	(LM_BASE[63:12]<<12) + 0x000000FFF
8KiB	(LM_BASE[63:13]<<13)	(LM_BASE[63:13]<<13) + 0x000001FFF
16KiB	(LM_BASE[63:14]<<14)	(LM_BASE[63:14]<<14) + 0x000003FFF
32KiB	(LM_BASE[63:15]<<15)	(LM_BASE[63:15]<<15) + 0x000007FFF
64KiB	(LM_BASE[63:16]<<16)	(LM_BASE[63:16]<<16) + 0x00000FFFF
128KiB	(LM_BASE[63:17]<<17)	(LM_BASE[63:17]<<17) + 0x00001FFFF
256KiB	(LM_BASE[63:18]<<18)	(LM_BASE[63:18]<<18) + 0x00003FFFF
512KiB	(LM_BASE[63:19]<<19)	(LM_BASE[63:19]<<19) + 0x00007FFFF
1MiB	(LM_BASE[63:20]<<20)	(LM_BASE[63:20]<<20) + 0x0000FFFFF
2MiB	(LM_BASE[63:21]<<21)	(LM_BASE[63:21]<<21) + 0x0001FFFFF
4MiB	(LM_BASE[63:22]<<22)	(LM_BASE[63:22]<<22) + 0x0003FFFFF

Continued on next page...

Table 45: (continued)

LM Size	Start	End
8MiB	(LM_BASE[63:23]<<23)	(LM_BASE[63:23]<<23) + 0x0007FFFFF
16MiB	(LM_BASE[63:24]<<24)	(LM_BASE[63:24]<<24) + 0x000FFFFFF

7.4 Local Memory Usage Constraints

Local memories are optimized for access latency. As a result, the design imposes the following usage restrictions:

- The addresses of VA and PA should be the same. Otherwise, UNPREDICTABLE behavior might happen.
- Accesses to the local memory are speculative. Devices with side effects on reads should not be mapped to this region.

7.5 Local Memory Interface

The local memory interface could be configured as SRAM or AHB-Lite through the **Local Memory Interface** option. If "ram" is selected, the SRAM-style interface will be configured. If "ahb-lite" is selected, the AHB-Lite interface will be used.

Table 46 shows the possible transactions of the AHB-Lite interface used by the Local Memory Interfaces.

Table 47 and Table 48 summarize the possible `HPROT` combinations on AHB-lite interfaces for instruction/data local memories.

Table 46: Possible AHB-Lite Transactions Used by Local Memory Interfaces

Request Types	Transaction Types
Write transfers	SINGLE DOUBLE WORD SINGLE WORD SINGLE HALF WORD SINGLE BYTE
Read transfers	SINGE DOUBLE WORD

Table 47: Instruction Local Memory Protection Control Signal

ILM_HPROT[3] Cacheable	ILM_HPROT[2] Bufferable	ILM_HPROT[1] Privileged	ILM_HPROT[0] Data/Instruction	Description
1	0	0	0	User instruction fetch
		0	1	User data access
		1	0	Privileged instruction fetch
		1	1	Privileged data access

Table 48: Data Local Memory Protection Control Signal

DLM_HPROT[3] Cacheable	DLM_HPROT[2] Bufferable	DLM_HPROT[1] Privileged	DLM_HPROT[0] Data/Instruction	Description
1	0	0	1	User data access
		1	1	Privileged data access

8 Local Memory Slave Port

8.1 Introduction

The LM slave port enables external bus masters to access the local memories of the processor. When an address exceeds ILM/DLM size, the higher address bits are ignored by the slave port. The `slv_huser` signal of the LM slave port interface selects which local memory to access:

Table 49: Local Memory Slave Port Selection

slv_huser[0]	Selection
0	ILM
1	DLM

The LM slave port supports all kinds of AHB burst transactions and contains a four-entry buffer for burst accesses or write accesses. Therefore, write transfers might temporarily be stored in the buffer. The LM slave port performs prefetch on burst read transfers to shorten the total wait cycles of burst transfers.

Slave port accesses have lower priority than load/store operations and instruction fetches. But when a slave port access is not granted within 4 cycles, the access is granted the highest priority to avoid starvation.

Note that the processor does not include logics to guarantee atomicity of atomic instructions accessing the LM address space when external masters access the same location through the LM slave port, nor does it provide the protection feature on LM slave port.

8.2 Latency of Transfer

The table below summarizes the minimum latency of transfers accessing the LM slave port. For read transfers, the latency will be larger than the listed number if the request of the LM slave port is not granted by the processor instantly. For write transfers, the latency will be larger than the listed number when the internal 4-entry buffer is full.

Table 50: Local Memory Slave Port Transfer Latency

Access Type	Minimal Latency
Single Read	5
Single Write	1

Continued on next page...

Table 50: (continued)

Access Type	Minimal Latency
Burst Read (N Beats)	N+4
Burst Write (N Beats)	N

8.3 Basic Transfer

The waveform below illustrates the best case of Single-read accesses with 4 cycle wait states and Single-write accesses with no wait state. Note that `BUS_CLK` is a pseudo-clock, please see Section 4.2 for more information.

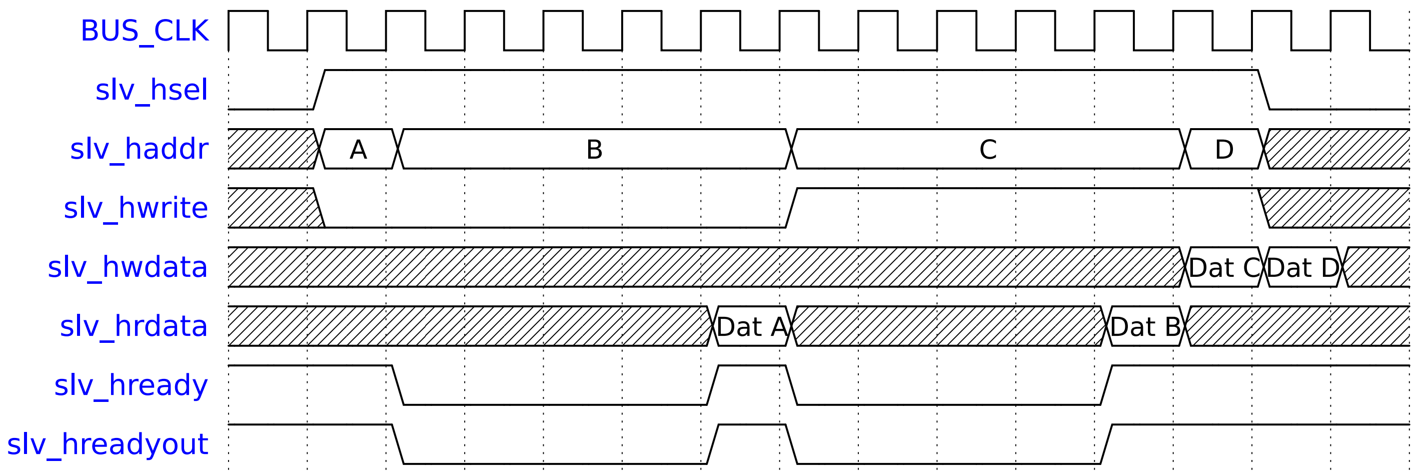


Figure 11: Single Accesses in the Local Memory Slave Port

8.4 Burst Transfer

Figure 12 and Figure 13 illustrate a 16-beat incremental burst read access and 16-beat wrapped burst write access. Note that extra wait states may be inserted if the data width of the LM slave port is not equal to the data width of the local memory.

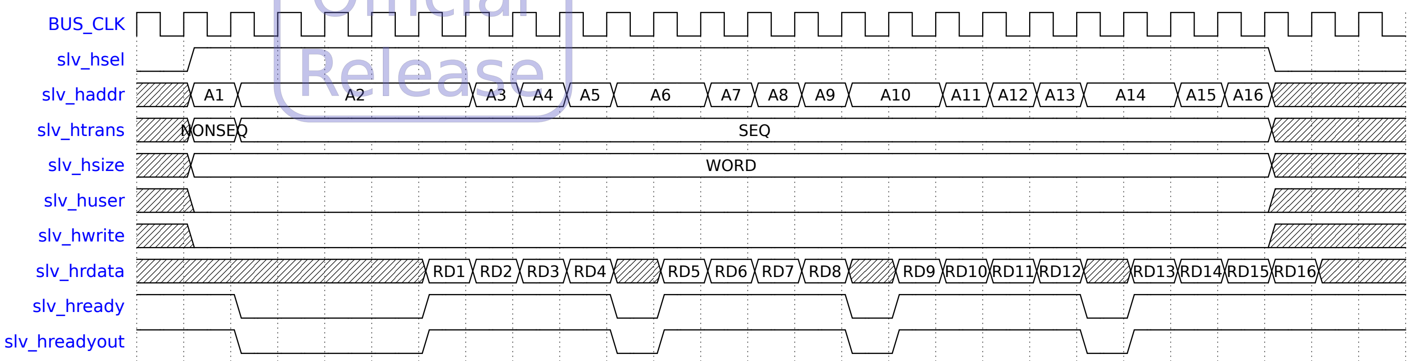


Figure 12: Burst Read Access in the Local Memory Slave Port

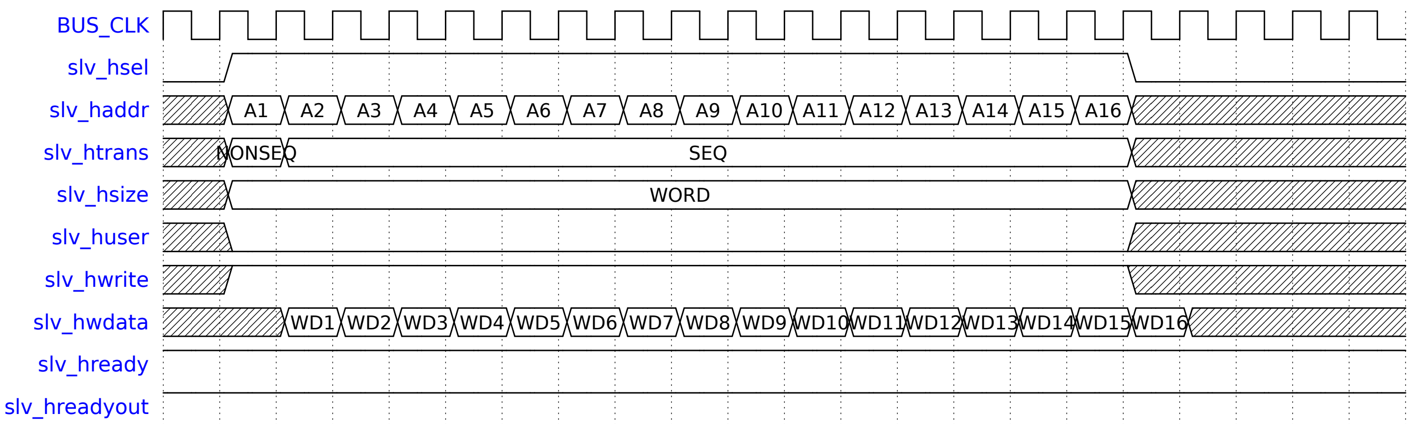


Figure 13: Burst Write Access in the Local Memory Slave Port

8.5 Support for Soft Error Protection

The LM slave port would return bus errors to bus masters as well as trigger local interrupts to NX25(F) when a uncorrectable ECC error or parity error is detected.

The behavior of ECC logic for local memories is controlled by `milmb.ECCEN/mdlmb.ECCEN`. The encoding for errors encountered through accesses from the LM slave port is summarized in the table below.

Correctable ECC errors only trigger local interrupts when `ECCEN` is equal to 3. Uncorrectable ECC errors would trigger local interrupts when `ECCEN` is equal to 2 or 3. The triggering of local interrupts is controlled by `mie.IMECCI` and the interrupt status is reported in `mip.IMECCI`. See Section 15.3.5 and Section 15.3.11 for details.

Data returned through the LM slave port is the ECC corrected version. For uncorrectable ECC errors, bus errors are reported when `ECCEN` is equal to 2 or 3.

ECCEN	Meaning	Data Returned
0	Disable parity/ECC	Uncorrected data
1	Reserved	Reserved
2	Generate local interrupts only on uncorrectable parity/ECC errors	Corrected data or bus errors
3	Generate local interrupts on any type of parity/ECC errors	Corrected data or bus errors

Extra wait states are needed when the size of a write transfer differs from the data bus width of the local memory.

Figure 14 demonstrates an example of write transfers consisting of various sizes with ECC.

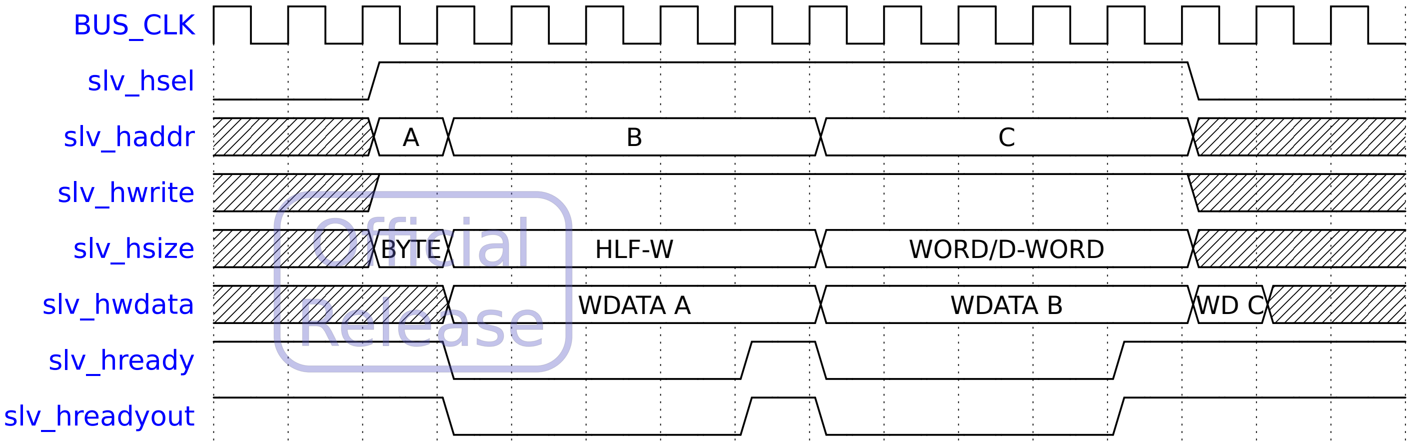


Figure 14: Example of Write Transfers Consisting of Various Sizes with ECC

8.6 Local Memory Slave Port Operation Under WFI Mode

When NX25(F) is in WFI mode, LM data may still need to be transferred through the slave port. The accessibility of LM is controlled by `lm_clk`, the only clock source for the slave port. Whether `core_clk` is gated or not, LM is accessible when `lm_clk` is active and `slv_reset_n` is de-asserted.

To gate `lm_clk` in WFI mode as well, the following conditions should be met:

- The core is in WFI mode.
- There are no more outstanding transfers in the LM slave port.

8.7 Local Memory Initialization

In some applications, data/instructions are loaded to LM before the core fetches the first instruction. To load data/instructions, the correct de-assertion sequence of `core_reset_n` and `slv_reset_n` is essential.

To make the LM slave port accessible when either the core or the external bus is under reset, the LM uses a merged reset signal derived from both `core_reset_n` and `slv_reset_n`. To avoid glitch on this merged reset signal, the reset sequence in the system should meet the following conditions:

- `slv_reset_n` should be de-asserted before `core_reset_n` is de-asserted.
- `core_reset_n` and `slv_reset_n` never go in the opposite direction at the same time (one goes high while the other goes low).

Between these two de-assertions, data/instructions can be loaded to LM through the slave port with `lm_clk` being active. The length of this period depends on application requirements and is controlled by the system.

`core_reset_n` and `slv_reset_n` are also the reset sources for local RAM modules. The local RAM modules are in reset state when both `core_reset_n` and `slv_reset_n` are asserted simultaneously. When one of these two signals is de-asserted, local RAM modules will exit the reset state. Moreover, a synchronizer is added for generating the internal reset signal from `core_reset_n` and `slv_reset_n`. It leads to a 2-cycle latency for the internal reset signal to be de-asserted after `core_reset_n` or `slv_reset_n` is de-asserted.

9 Caches

9.1 Introduction

The processor has two caches, the instruction cache and the data cache. The cache sizes of both are configurable.

The cache organization information can be collected from the `micm_cfg` register for the instruction cache and the `mdcm_cfg` register for the data cache. The configuration choices are listed below, and the format of the configuration registers can be found in Section 15.6.1 and Section 15.6.2.

Table 51: Configuration Choices for the Instruction Cache

Items	**Field Name (micm_cfg)**	**Choices**
Cache lines per way	ISET	64, 128, 256, 512, 1024, 2048
Ways	IWAY	Direct-mapped, 2-way , 4-way
Line size (bytes)	ISZ	32

Table 52: Configuration Choices for the Data Cache

Items	**Field Name (mdcm_cfg)**	**Choices**
Cache lines per way	DSET	64, 128, 256, 512, 1024, 2048
Ways	DWAY	Direct-mapped, 2-way , 4-way
Line size (bytes)	DSZ	32

Total cache size = Cache lines per way $\times$ Ways $\times$ Line size. 2-way and 4-way caches implement random or pseudo-LRU replacement policy. The replacement policy for direct-mapped caches is irrelevant.

9.2 Cache Access Latency

The access latency of the instruction cache and the data cache is listed below.

Table 53: Access Latency of the Instruction Cache

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
Fetch from I-Cache (hit)	1	2
Fetch from I-Cache (miss)	6*	7*

- Note: The calculation of latency should take system delay into consideration.

Table 54: Access Latency of the Data Cache

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
Load word/dword from D-Cache (hit)	1	2
Load word/dword from D-Cache (miss)	5*	7*
Load byte/halfword from D-Cache (hit)	1	3
Load byte/halfword from D-Cache (miss)	5*	7*

- Note: The calculation of latency should take system delay into consideration.

9.3 I-Cache Fill Operation

The instruction cache fill operation starts when a cacheable line is not in the I-Cache. A burst read request for the missed cache line is always sent first to the system bus to minimize the miss latency.

The fill operation may be aborted by system bus errors. A precise instruction access fault is triggered for the instruction fetch causing the cache miss operation if the error is on the critical word. If the error occurs on non-critical words, the fill operation will be canceled and the missed line will not be installed into I-Cache. Instruction fetches before non-critical error words will not be affected since they have received the required data.

In Debug Mode, instruction fetches will not affect I-Cache contents, and all I-Cache misses will not cause cache replacements.

9.4 D-Cache Fill Operations

The D-Cache fill operation starts when a cacheable line is not in the D-Cache. A burst read request for the missed cache line is always sent first to the system bus to minimize the miss latency. The read request will be followed by a burst write request if cache eviction is required.

The fill operation may be aborted by system bus errors. A precise load/store access fault is triggered for the load/store instruction causing the cache miss operation if the error is for the critical word. If the error occurs on non-critical words, the fill operation will be canceled and the missed line will not be installed into D-Cache. Load instructions will not be affected by these errors since they have received the required data (the critical words). Store instructions will send a single bus request to write the data directly to the bus.

In Debug Mode, load/store instructions will minimally affect D-Cache contents. All cache misses will not cause cache replacements, and only dirty bits may be affected by accesses to cache lines that are already in D-Cache.

9.5 D-Cache Eviction Operations

A burst write request will be sent to the system bus if a dirty line is evicted out of D-Cache. An imprecise bus-write error exception is triggered if the burst write request encounters system bus errors.

9.6 FENCE/FENCE.I Operations

`FENCE/FENCE.I` instructions may affect the cache. The behavior of `FENCE/FENCE.I` is summarized in Table 55.

Table 55: Effects of FENCE/FENCE.I Instructions

Cache	FENCE	FENCE.I
I-Cache	None	Invalidate all cache lines
D-Cache	None	Write back all cache lines

9.7 CCTL Operations

CCTL operations provide direct control to manipulate instruction or data caches (cache maintenance operations). They are invoked by writing CCTL commands to the `mcctlcommand` CSR register. The operations can be grouped into two main types, virtual-address (VA) based or index (IX) based, and they are summarized in Table 56. These two addressing types affect how cache lines are specified for CCTL operations, and how the content of `mcctlbeginaddr` are interpreted.

Table 56: Addressing Type of CCTL Commands

Type	Usage
IX	Use the content of `mcctlbeginaddr` register directly as a (Way, Index) or (Way, Index, Double-Word) pair/tuple to specify a cache line in the cache without going through any translation mechanism. The format is defined in Table 57.
VA	Use the content of `mcctlbeginaddr` register as a virtual address to access the cache. The virtual address has to go through the same address translation mechanism in the processor pipeline as the address of regular load/store instructions for D-Cache or instruction fetches for I-Cache. The specified operation is performed only if the addressed cache line is in the corresponding cache.

Table 57: Index Format for Index Type of CCTL Operations

Field	Bit Position	Description
OFFSET	`mcctlbeginaddr[A-1:3]`	A=log_2(#Double-Words in a Cache Line) + 3
INDEX	`mcctlbeginaddr[B-1:A]`	B=log_2(Cache Size / #Ways)
WAY	`mcctlbeginaddr[C-1:B]`	C=Ceiling(log_2(Cache Size))

The following diagram shows an example of `mcctlbeginaddr` for the index type of CCTL operations, assuming that cache is 4-way, 32-KiB with 32-byte cache line.

63 – 15	14 – 13	12 – 5	4 – 3	2 – 0
0	WAY	INDEX	OFFSET	0

All available CCTL operations are summarized in Table 88. Their detailed definitions are grouped and described in the following categories.

1. **Invalidating cache blocks** (L1D_VA_INVAL, L1I_VA_INVAL, L1D_IX_INVAL, L1I_IX_INVAL)
   These operations invalidate the specified cache lines. Locked cache lines are unlocked and invalidated.

2. **Writing back cache blocks** (L1D_VA_WB, L1D_IX_WB, L1D_WB_ALL)
   These operations write the data of the specified cache lines back to the system memory, if the specified cache lines are present in the cache with dirty states. The specified cache lines will still be kept in the cache and locked cache lines remain locked.

3. **Writing back & invalidating cache blocks** (L1D_VA_WBINVAL, L1D_IX_WBINVAL, L1D_WBINVAL_ALL)
   These operations write the data of the specified cache lines back to the system memory, if the specified cache lines are present in the cache with dirty states. Then the specified cache lines will be invalidated as long as they are valid in the cache, regardless of whether their states are dirty or locked.

4. **Filling and locking cache blocks** (L1D_VA_LOCK, L1I_VA_LOCK)
   These operations lock the specified cache lines in the cache. The specified cache lines are first brought into the cache if they are not already present in the cache, then the cache lines are locked by setting their lock states. It is not an error to lock an already locked line—the same line is just locked again.

   A locked cache line will not be replaced by the cache replacement policy on cache misses/fills. It can only be unlocked by the cache invalidate or unlock operations.

   The cache lines of a same cache-line set cannot be locked at the same time. That is, the maximum number of cache lines that can be locked is one less the associativity of the cache. These operations will abort when the number of locked cache lines in the specified cache-line set already reaches the maximum.

   The status of these operations are written to the `mcctldata` register:

   - A value of 1 indicates that the lock operation finished successfully;
   - A value of 0 indicates that the lock operation aborted/failed.

5. **Unlocking cache blocks** (L1D_VA_UNLOCK, L1I_VA_UNLOCK)
   These operations clear the lock state of the specified cache lines if the specified cache lines are present in the cache.

6. **Reading tag data from caches** (L1D_IX_RTAG, L1I_IX_RTAG)
   These operations read the contents of the tag part of the target cache line into the `mcctldata` register. The format of the tag data in `mcctldata` is defined in Section 15.10.10. The target cache line is specified in the `mcctlbeginaddr` register by its way and index information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in `mcache_ctl` to read the corresponding ECC /Parity codes in the tag RAM to `mecc_code`.

7. **Reading data from caches** (L1D_IX_RDATA, L1I_IX_RDATA)
   These operations read a 8-byte data from a cache line into the `mcctldata` register. The target cache line is specified in the `mcctlbeginaddr` register by its way, index, and double word information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in `mcache_ctl` to read the corresponding ECC /Parity codes in the data RAM to `mecc_code`.

8. **Writing tag data to caches** (L1D_IX_WTAG, L1I_IX_WTAG)
   These operations write the contents of the `mcctldata` register to the tag part of the target cache line. The format of the tag data in `mcctldata` is defined in Section 15.10.10. The target cache line is specified in the `mcctlbeginaddr` register by its way and index information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in `mcache_ctl` to write the `mecc_code` ECC /Parity code to the corresponding tag RAM.

9. **Writing data to caches** (L1D_IX_WDATA, L1I_IX_WDATA)
   These operations write a 8-byte data in the `mcctldata` register into the target cache line. The target cache line is specified in the `mcctlbeginaddr` register by its way, index, and double word information. Additionally, these operations observe DC_RWECC/IC_RWECC settings in `mcache_ctl` to write the `mecc_code` ECC /Parity code to the corresponding data RAM.

10. **Invalidating all cache blocks** (L1D_INVAL_ALL)
    This operation invalidates all valid lines of D-Cache. Locked cache lines are unlocked and invalidated.

11. **Writing back all cache blocks** (L1D_WB_ALL)
    This operation writes the data of all dirty cache lines of D-Cache back to the system memory. All cache lines will still remain in the D-Cache and locked lines remain locked.

12. **Writing back & invalidating all cache blocks** (L1D_WBINVAL_ALL)
    This operation writes the data of all dirty cache lines of D-Cache back to the system memory and all valid cache lines will be invalidated, including locked and/or clean cache lines.

9.8 User CCTL Operations

Four CCTL operations (Table 58) are available to User-mode software under the control of the `mcache_ctl.CCTL_SUEN` control bit. These operations are triggered in the User-mode by accessing `ucctlbeginaddr` and `ucctlcommand` registers while `mcache_ctl.CCTL_SUEN` controls access permission to these registers. When `CCTL_SUEN` is 0, accessing them in User mode would cause illegal instruction exceptions. It should be set to 1 to enable User CCTL operations.

Table 58: User CCTL Operations

Value		Command	Type	Exception Entry
0	0b00_000	L1D_VA_INVAL	VA	Store related Fault
1	0b00_001	L1D_VA_WB	VA	Store related Fault
2	0b00_010	L1D_VA_WBINVAL	VA	Store related Fault
8	0b01_000	L1I_VA_INVAL	VA	Store related Fault

9.9 Interruption of CCTL Operations

All CCTL operations are interruptible. For CCTL operations that process a single cache line, no operations are performed upon interruption.

For CCTL operations that process the entire cache (L1D_WB_ALL L1D_WBINVAL_ALL L1D_INVAL_ALL), they could be interrupted in the middle of operations. Next execution of the same instruction will start from the start, instead of from the point of interruption. To resume from the point of interruption, a for-loop should be iterate over the entire cache using L1D_IX_WB, L1D_IX_WBINVAL or L1D_IX_INVAL, respectively. But note that the D-Cache states could have been disturbed by the interrupt handler and the state of the D-Cache at the end of the for-loop will not be guaranteed to be entirely clean as the end of the respective CCTL operation on the whole D-Cache.

Additionally, note that CCTL operations take parameters from multiple CSR registers, thus they are inherently not atomic. To allow CCTL operations to be used in multiple privilege levels and/or non-interrupt code, interrupt handlers as well as higher privilege level software should backup the content of CCTL CSR registers with interrupts disabled before using them, and restore their values afterwards.

10 Bus Interface Unit

10.1 Introduction

The bus interface unit (BIU) is responsible for off-CPU accesses which include system memory accesses and memory-mapped register accesses in devices. This unit supports both AHB and AXI bus protocols.

10.2 Block Diagram

BIU includes an arbiter, a command FIFO, a write-data FIFO, a response FIFO, and a bus master.

The following figure shows the block diagram of BIU. Requests for the external bus can come from fetch, load, and store accesses to memory.

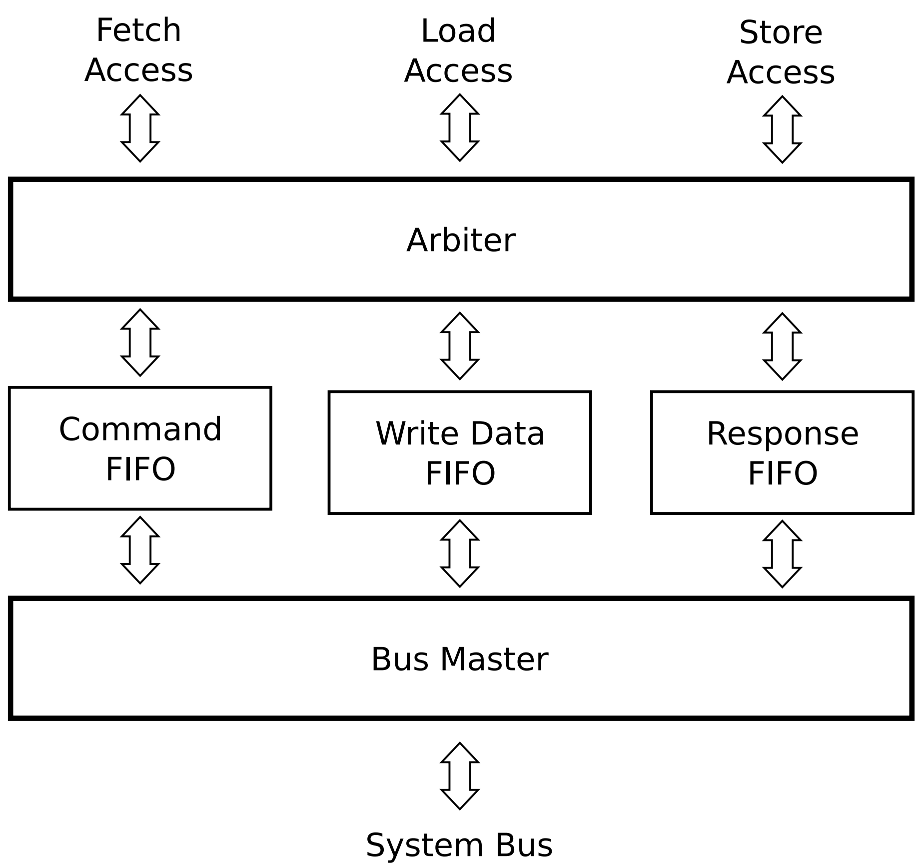


Figure 15: BIU Block Diagram

10.3 Optional BIU Two-Port Structure

BIU provides a two-port configuration option for AHB bus (AHBx2) and AXI bus (AXIx2). With this option, instruction and data accesses are split into separate ports: the instruction bus and the data bus. All instruction fetches go through the instruction bus while all requests from load/store and page table walker accesses go through the data bus.

The debug module Section 20 needs to be accessed by instruction fetch and data accesses. When this option is configured, please make sure that both ports could access the address space of the debug module.

If there are regions of address space which both ports (the instruction bus and the data bus) have shared accesses and the bus type is AHB (e.g., debug module as a AHB slave), please note the following usage limitations:

- The device store-load sequence with data dependency (read-after-write hazard) might cause the bus to hang if the priority of the instruction bus is always higher than the data bus. This is because the latter load will be replayed upon detection of such hazards to guarantee that the earlier store is finished. Replaying means the load instruction will need to be re-fetched again from memory. This often occurs sooner than when the store instruction needs the data bus. As a result, the store instruction will be blocked waiting for instruction fetches to yield, but the load fetched always detects the read-after-write hazard and keeps instruction bus busy.

  It is recommended that the arbiter of the common bus should at least periodically grant accesses to the data bus accesses.

- Atomic instruction sequences involving LR-SC pairs should not lock the common bus, as the bus will be locked by LR instructions and only unlocked by SC instructions, and the locking will preclude instruction bus port to access the common bus and get SC instructions to unlock the bus.

- AMO instructions encountering ECC errors might cause the processor to hang. AMO instructions will lock the bus on the read operation until the write operation is done. When ECC errors are detected after the bus is locked, AMO instructions will be replayed for correctable errors without unlocking the bus. Replaying means that the processor will re-fetch and re-execute the AMO instruction. However, the bus is locked, precludes the instruction bus port to be granted accesses, and causes the processor to hang.

  It is recommended to avoid AMO instructions when the two bus ports are eventually connected to a common AHB bus structure.

10.4 Supported Transaction Types

Table 59 and Table 60 summarize the transactions that the processor uses to access the AHB and AXI buses.

Table 59: Possible AHB Transactions

Request Types	Transaction Types
Basic transfers	SINGLE DOUBLEWORD SINGLE WORD SINGLE HALFWORD SINGLE BYTE
Additional transfers when caches are configured	WRAP4 DOUBLEWORD

Table 60: Possible AXI Transactions

Request Types	AxBURST	AxLEN	AxSIZE
Basic transfers	INCR	0	DOUBLEWORD WORD HALFWORD BYTE
Additional transfers when caches are configured	WRAP	3	DOUBLEWORD

10.5 Atomic Operations

Atomic operations are implemented by locking the bus using the `HLOCK` signal when AHB is supported. The `HLOCK` signal is asserted to lock the AHB bus once the `LR` instruction is executed. BIU will hold the lock until the `SC` instruction is executed or the internal lock is cleared.

The Exclusive access mechanism is used for implementing the atomic operations to the AXI bus: `ARLOCK`==1 for load misses caused by `LR` instructions and `AWLOCK`==1 for store misses caused by `SC` instructions.

Please see the usage descriptions for `LR` and `SC` instructions in the *RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2*.

10.6 Low Latency AHB Access Mode

The low latency access option is designed to improve the bus access latency under low clock frequency conditions.

This mode comes with a couple of restrictions:

1. Only the AHB bus is supported.
2. The bus must be synchronous although the bus clock could be configured to be slower (N:1 clock ratio) in runtime to reduce power consumption. The timing constraint assumes 1:1 clock ratio with respect to `core_clk` to support the maximal frequency during synthesis.
3. Cycle time is sacrificed in order to shorten the access latency. Bus signals are used directly without flops so internal critical paths will appear in the SoC timing report — 1/2 of the bus cycle time should be reserved for internal paths of BIU.

10.7 Number of Outstanding AXI Transactions

The maximum number of outstanding transactions on AXI AR channel is 3. The maximum number of outstanding transactions on AXI AW channel is 6.

And the maximum numbers for each transaction are as below:

- Store transaction: 6
- Load transaction: 1
- Instruction fetch: 1 (I-Cache RAM size is zero) or 2

Note

AE350 uses ATCBMC300 as the platform level bus interconnection which instances as vc_bmcx, vc_mstmux, etc. ATCBMC300 limits the number of outstanding requests for a downstream port (ATCBMC300_SLVx_FIFO_DEPTH, x=1~31), which can be greater than one only if the connected AXI slave returns responses in order regardless of AXI IDs. Therefore, The ATCBMC300_SLVx_FIFO_DEPTH is set to one inside ae350_bus_connector to the connected AXI slaves for maximal flexibility. Accordingly, the overall maximal outstanding performance is lower than that shown above. It is recommended to search for other AXI bus interconnection IPs if this limitation violates the customer's requirement. Please see document "AndeShape_ATCBMC300_DS119" for details.

10.8 AXI ID Value Assignment

The width of AXI IDs are 4 bits wide and the table below lists their value assignments.

Table 61: AXI ID Assignments on the AXI Interface

Value	Description
0x0	Transactions caused by load instructions or by cacheable stores that miss D-Cache
0x2	Transactions caused by instruction fetches
0x3	Transactions caused by store instructions to noncacheable/write-through regions and by D-Cache evictions
Others	Unused

10.9 AXI AWCACHE/ARCACHE Description

Supported memory types of AXI AWCACHE/ARCACHE are listed in the table below.

Table 62: AXI AWCACHE/ARCACHE Values

AWCACHE/ARCACHE	Description
0000	Device non-buffer
0110	Write-through read allocate
1111	Write-back R/W allocate

10.10 AXI AWPROT/ARPROT Description

- AXPROT[0]: Normal or privileged
  - 1: privileged access (non user mode)
  - 0: normal access (user mode)
- AXPROT[1]: Secure or non-secure
  - 1: non-secure access
- AXPROT[2]: Instruction or data
  - 0: data access
  - 1: instruction access

10.11 AHB HPROT Description

- HPROT[0]: opcode fetch or data access
  - 0: opcode fetch
  - 1: data access
- HPROT[1]: a machine(privileged) mode access or user mode access.
  - 0: user mode access
  - 1: machine(privileged) mode access
- HPROT[3:2]: cacheable or bufferable access.
  - 00: device region access
  - 10: normal memory, write through region access
  - 11: normal memory, write back region access

11 Trap

11.1 Introduction

According to the RISC-V Privileged Architecture, a trap is a control flow change of normal instruction execution caused by an interrupt or an exception. An interrupt is a control flow change event initiated by an external source. An exception is a control flow change event generated as a by-product of instruction execution. When a trap happens, the processor stops processing the current flow of instructions, disables interrupts, saves enough states for later resumption, and starts executing a trap handler.

Interrupts can be local or external. The external interrupts are global interrupts that are arbitrated externally by a platform level interrupt controller (PLIC) and the selected external interrupt joins the rest of local interrupts for arbitration to take a trap.

Exceptions can be precise or imprecise. The instruction causing precise exceptions and all its subsequent instructions in the program order will not have affected the architectural state when precise exceptions are triggered. Furthermore, the events that cause these precise exceptions have to be precisely attributed to the causing instruction. The value of `mcause` register will be greater than zero for precise exceptions. Exceptions not meeting these criteria can only be imprecise and they are delivered as local interrupts (`mcause` < 0) instead. That is, the standard RISC-V privileged architecture exceptions are only triggered for precise exceptions, and local interrupts are triggered for imprecise exceptions.

For precise exceptions, `mepc` is the PC of the faulting instruction. For imprecise exceptions, `mepc` is pointing to the interrupted instruction. Regardless of preciseness of exceptions, `mtval` records the effective faulting address for exceptions related to memory operations.

11.2 Interrupt

The processor provides three interrupt inputs: Timer interrupt, software interrupt, and external interrupt. Timer interrupts and software interrupts are local interrupts in a RISC-V platform, which means each processor in the platform receives its own timer/software interrupts. External interrupts are global interrupts in a RISC-V platform, which means they are shared by all processors in a RISC-V platform. External interrupts are arbitrated and distributed by a platform-level interrupt controller (PLIC) to a processor. Each external interrupt source can be assigned its own priority, and each interrupt target (i.e., RISC-V processors) could select which external interrupt sources it would handle. PLIC routes the highest priority interrupt source to the target processor. See Section 18 for more descriptions on PLIC.

11.2.1 Additional Local Interrupts

In addition to external interrupts, the processor may generate internal interrupts for the following events (imprecise exceptions):

- Local memory slave port parity/ ECC error (See Section 8.5, mie.IMECCI and mip.IMECCI)
- Bus read/write transaction error (See mie.BWEI and mip.BWEI)
  - PMP check errors can be reported as bus read/write/transaction errors if they are only detected after the first micro-operation.
- Performance monitor overflow (See mie.PMOVI and mip.PMOVI)

Note

Parity/ ECC and bus errors could be either precise or imprecise. It all depends on whether the pipeline can attribute the errors to the faulting instruction and preserve the architectural states from being affected by the faulting instruction and its subsequent instructions. If these errors can be precise exceptions, access faults (precise exceptions) are triggered. Otherwise, local interrupts (imprecise exceptions) will be triggered. See the next section for more details and see the tables in Section 15.3.13 for how a trap handler can distinguish between them.

11.2.2 Interrupt Status and Masking

The `mip` CSR contains pending bits of these interrupts, with the `mie` CSR contains enable bits of the respective interrupts. The processor can selectively enable interrupts by manipulating the `mie` CSR, or globally disable interrupts by clearing the `mstatus.MIE` bit.

11.2.3 Interrupt Priority

When multiple interrupts are taken at the same time, they are handled under the following order:

Interrupt Handling Priority
M-mode performance monitor overflow interrupt
M-mode bus read/write transaction error interrupt
M-mode imprecise ECC error interrupt
M-mode external interrupt (MEI)
M-mode software interrupt (MSI)
M-mode timer interrupt (MTI)
U-mode external interrupt (UEI)
U-mode software interrupt (USI)

Continued on next page...

Interrupt Handling Priority
U-mode timer interrupt (UTI)

11.3 Exception

The processor implements the following (precise) exceptions (`mcause` > 0). See the tables in Section 15.3.13 (and Section 15.3.9) for how these exceptions can be identified by trap handlers.

- Instruction address misaligned exceptions
  - Jump to misaligned addresses
- Instruction access faults
  - Bus errors caused by instruction fetches
  - Uncorrectable ECC errors when fetching trap handlers under the vector PLIC mode
- Illegal instructions
  - Unsupported instructions
  - Privileged instructions
  - Accessing non-existent CSRs
  - Accessing privileged CSRs
  - Writing to read-only CSRs
  - Executing Andes-specific instructions in the RISC-V compatibility mode (`mmisc_ctl.RVCOMPM` == 1).
  - ACE instructions with reserved sub-opcode or register index
- Breakpoint exceptions
- Load address misaligned exceptions
- Load access faults
  - Bus errors caused by load instructions
  - ECC errors caused by load instructions
- Store/AMO address misaligned exceptions
- Store/AMO access faults
  - ECC errors caused by store instructions
- Environment calls
- Stack overflow/underflow exceptions with StackSafe supported
- ACE disabled exceptions
  - Executing ACE instructions without enabling ACE context (`mmisc_ctl.ACES` == 0)
- ACE exceptions
  - Bus errors caused by memory accesses
  - Misaligned or out-of-range memory address exceptions
  - Zero length vector exceptions

– Custom-defined exceptions

Some events (for example, parity/ECC, and bus errors) listed above may cause imprecise exceptions in some circumstances instead. Imprecise exceptions are delivered through local interrupts (`mcause` < 0) instead of the standard RISC-V exceptions (`mcause` > 0). It all depends on the ability of the pipeline to attribute the errors to the faulting instruction and keep the architectural state clean from being polluted by the faulting instruction and all of its subsequent instructions. For example, bus read errors on non-critical word cannot be attributed to any of the executed instructions and will be imprecise.

Most errors related to address checks are precise, unless the instruction is split into micro-operations and the error is found not on the first micro-operation. For example, PMP check errors for the second micro-operation of a misaligned memory accesses.

11.4 Trap Handling

11.4.1 Entering the Trap Handler

When a trap occurs, the following operations are applied:

- `mepc` is set to the current program counter.
- `mstatus` is updated.
  - The `MPP` field is set to the current privilege mode.
  - The `MPIE` field is set to the `MIE` field.
  - The `MIE` field is set to 0.
- `mcause` is updated.
- `mtval` is updated on any of address-misaligned or access-fault exceptions.
- The privilege mode is changed to M-mode.
- When `mmisc_ctl.VEC_PLIC` is 0, the program counter is set to the address specified by `mtvec`.
- When `mmisc_ctl.VEC_PLIC` is 1, the `mtvec` register will be the base address register of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.
  - `mtvec[0]` is for exceptions and non-external local interrupts. For these traps, the `mcause` register records the trap type based on RISC-V definitions.
  - `mtvec[i]` is for external PLIC interrupt source *i* triggered through the `mip.MEIP` pending condition.
  - `mtvec[1024+i]` is for external PLIC interrupt source *i* triggered through
    * the `mip.UEIP` pending condition when `mideleg.UEI` == 0 for M/U systems.
  - For external PLIC interrupts, the `mcause` register records the interrupt source ID.

  - The upper 32-bit address is equal to `mtvec[63:32]`. The RISC-V architecture defines a two-level stack of interrupt enable bits and privilege modes. To support nested traps, the trap handler should back up trap handling CSRs and enable the interrupt enable bit.

11.4.2 Returning from the Trap Handler

After handling a trap, the `MRET` instruction can be executed for returning to the instruction and the privilege context before the trap happened. Alternatively, the trap handler could assign new PC, privilege level and/or interrupt enable status to `mepc`, `mstatus.MPP` and `mstatus.MPIE` before `MRET`. Specifically, the following operations take place when an `MRET` instruction is executed:

- The program counter is set to `mepc`
- The privilege mode is set to `mstatus.MPP`
- `mstatus` is updated
  - The `MPP` field is set to U-mode (or M-mode if U-mode is not supported)
  - The `MIE` field is set to the `MPIE` field
  - The `MPIE` field is set to 1

12 Reset and Non-Maskable Interrupts

12.1 Reset

When the core_reset_n input signal of the processor is deasserted, the following operations are applied:

- CSRs are set to their reset values.
- All integer registers (listed in Table 40) are set to zero.
- BTB is initialized.
- Program execution starts with the address specified by the reset_vector input signal.

12.2 Non-Maskable Interrupts

Non-maskable interrupts (NMIs) are intended for handling hardware error conditions and are assumed to be non-resumable. They are triggered through the nmi input signal. The rising edge of the signal causes an immediate jump to an address stored in the mnvec register and transition of the privilege level to M-mode, regardless of the state of a hart's interrupt enable bit.

The following operations are applied when an NMI is taken:

- The mepc register is written with the address of the next instruction when the NMI was taken.
- The mcause register is set to 1, indicating that NMI is caused by the reset_vector input signal.
- The mstatus.MPP field records the privilege mode before NMI was taken.
- The mstatus.MPIE field is set to the value of mstatus.*x*IE before NMI was taken. The "x" is the active privilege mode before the NMI was taken.
- The mstatus.MIE field is set to 0.

13 Power Management

13.1 Wait-For-Interrupt Mode

The processor enters the wait-for-interrupt (WFI) mode with the `WFI` instruction for reducing power consumption, and clock-gating or power-gating of the processor should only happen when the processor is in the WFI mode.

Upon execution of the `WFI` instruction, the processor stops all activities and asserts the `core_wfi_mode` output signal to indicate that this processor is in the WFI mode.

Once in the WFI mode, memory transactions that are started before the execution of `WFI` are guaranteed to have been completed, all transient states of memory handling are flushed and no new memory accesses will take place.

In this period, the `core_clk` and `bus_clk_en` input signals can be safely gated to reduce the power consumption or changed for frequency scaling. This is also the safe period to power-gate the processor and leave the I/D-Cache SRAMs entering the state retention mode. Please note that when `core_clk` is gated, the processor cannot respond to wake-up events. For the processor to be woken up, an external logic should be present to detect the desired wake-up event and resume `core_clk` first.

Slave port accesses are not affected by WFI mode. The availability just depends on whether `lm_clk` is active. If slave port accesses are still needed in WFI mode, `lm_clk` should still be clocked while `core_clk` may be gated off.

`nmi`, `debugint` and interrupts defined in the `mip` CSR may cause the processor to leave the WFI mode.

The `nmi` or `debugint` signals cause the processor to leave the WFI mode unconditionally, as long as the core clock is toggling. The processor will resume and start to execute from the first instruction of NMI or debug-interrupt service routine.

All interrupts defined in the `mip` CSR may cause the processor to leave the WFI mode, depending on the setting of the `mie` CSR: interrupts disabled by the `mie` CSR will not be able to wake up the processor. However, the processor can be awoken by these interrupts regardless the value of the global interrupt enable bit (`mstatus.MIE`).

When the processor is awoken by a pending interrupt and `mstatus.MIE` is enabled, it will resume and start to execute from the corresponding interrupt service routine. When the processor is awoken by a pending interrupt and `mstatus.MIE` is disabled, it will resume and start to execute from the instruction after the WFI instruction.

Please note that the RISC-V ISA only defines the `WFI` instruction as a hint instruction. For portability, `WFI` instructions should not be assumed to always cause the processor to pause until interrupt arrives; they may be implemented as NOPs by other implementations and should be inside loops that exit when (`mie&mip`) is not zero.

13.2 Low Power Control

(This is an experimental feature in this release. Please contact Andes for more information.) Further power management could be achieved by more clock and power control mechanism, such as DVFS (Dynamic Voltage Frequency Scaling) and power domain on/off.

The processor also provides the System Management Unit (SMU) as a reference design to optimize power management by controlling the flow of power and clock changes. For power on/off management, the system is partitioned into an always-on domain and other controllable power domains. SMU, which resides in the always-on domain, takes care of the power control of other domains. In general, upon taking the power operation command, SMU shuts off the power of the target domain after this domain goes idle, and finally resumes the power of this domain after receiving any preset wakeup event.

SMU also provides the flow control of the clock gating of specific function units, such as processor cores. The clock control procedure is similar to that of power control. SMU takes the clock on/off command, shuts off the clock of the target function unit after checking the readiness of this function unit, and finally resumes the clock of this function unit after receiving any preset wakeup events.

14 Memory Subsystem Error Protection

14.1 Introduction

The processor includes support of soft-error protection for memory subsystems.

14.1.1 Memory Subsystem Error Protection Scheme

Two memory subsystem error protection schemes are supported:

- Parity
  - Memory error detection through even parity check
  - Single-bit error detection per byte
  - Each 8-bit data requires one extra bit to store the parity bit
  - Clean cache lines with parity errors detected can be corrected by invalidating the line.
- ECC
  - Single-Error-Correction, Double-Error-Detection (SEC-DED) ECC
  - Single-bit errors can be detected and corrected
  - Double-bit errors can be detected but may not be corrected
  - For Instruction Cache, each 32-bit data requires seven extra bits to store the ECC code
  - For Data Cache, each 64-bit data requires eight extra bits to store the ECC code
  - For Instruction local memory and Data local memory, each 64-bit data requires eight extra bits to store the ECC code
  - Clean cache lines with multi-bit errors detected can be corrected by invalidating the line.

14.1.2 Error-Protected Memory Subsystems

The memory subsystems protected by the Parity/ECC scheme include:

- Cache memories
  - Instruction caches (Tag RAM and Data RAM)
  - Data caches (Tag RAM and Data RAM)

- Local memories
  - ILM RAM
  - DLM RAM

14.1.3 Read-Modify-Write Operations

For data RAMs, the granularity of ECC protection is 64-bit double-word (eight bytes). The ECC code is computed and written to the data RAMs along with the data word.

To write narrower data (e.g., a byte or a halfword) into these RAMs, the design must read data from the RAM, merge the read data with the write data, and then generate the ECC code for the merged data before writing back the merged data and the ECC code. This process is referred to as read-modify-write operations and these operations are done automatically by hardware in the processor.

For parity protected RAMs, the unit of parity protection is one byte. The parity bits could be generated directly for all kinds of partial-word (byte and halfword) writes without the need for the read-modify-write operations.

The ECC protection scheme offers more protection at the cost of longer access latency for narrow data accesses. The parity protection scheme offers less protection but without the narrow data access performance affected.

14.2 Parity/ECC Control Modes

For each protected memory, a Parity/ECC enable control flag (`ECCEN`) is defined with three modes:

- Parity/ECC checking disabled
- Generating exceptions on uncorrectable Parity/ECC errors only
- Generating exceptions on all Parity/ECC errors

14.2.1 Parity/ECC Checking Disabled

The behavior of the Parity/ECC logic when Parity/ECC checking is disabled is:

- Reads from RAMs will not trigger the ECC circuit at all. The raw data is directly returned. But all writes to RAMs still update/regenerate Parity/ECC codes.
  - No exceptions would be generated.
  - No Parity/ECC-related registers will be updated.

- The design uses read-modify-write operations to store a partial word if the protection scheme is ECC.

14.2.2 Generating Exceptions on Uncorrectable Parity/ECC Errors Only

The behavior of the Parity/ECC logic under this mode is:

- Parity/ECC checking is enabled and all writes to RAMs update/regenerate Parity/ECC codes.
- The design gets the error-corrected data from RAMs.
- The design uses read-modify-write operations to store a partial word if the protection scheme is ECC.
- For accesses by the main processor pipeline:
  - No exception would be generated for correctable errors.
  - Exceptions would be generated for uncorrectable errors.
  - The related error reporting registers would be updated on all Parity/ECC errors.
- For accesses by the local memory slave port:
  - No exception would be generated for any detected Parity/ECC errors.
  - The standard bus error reporting mechanism is used to report uncorrectable errors for the slave port accesses.
  - The slave port triggers local interrupts to signal that uncorrectable errors are detected.

14.2.3 Generating Exceptions on All Parity/ECC Errors

The behavior of the Parity/ECC logic under this mode is:

- Parity/ECC checking is enabled and all writes to RAMs update/regenerate Parity/ECC codes.
- The design gets the error-corrected data from RAMs.
- The design uses read-modify-write operations to store a partial word if the protection scheme is ECC.
- For accesses by the main pipeline in the CPU core:
  - All detected Parity/ECC errors would generate exceptions.
  - Correctable errors will be corrected while the exceptions are triggered.
  - The related error reporting registers would be updated on all Parity/ECC errors.

- For accesses by the local memory slave port:
  - No exception would be generated for any detected Parity/ECC errors.
  - The standard bus error reporting mechanism is used to report uncorrectable errors for the slave port accesses.
  - The slave port triggers local interrupts to signal that uncorrectable errors are detected.

14.3 Behavior of Parity/ECC Error Exceptions

When Parity/ECC exceptions are triggered, the `mxstatus.DME` flag will be set, which will force the caches to be bypassed. The exception handler will thus not need to worry about tripping upon further Parity/ECC errors in caches. For I-Cache, since it is a clean cache and all ECC errors could be repaired, there is no need to set `mxstatus.IME` to bypass it and `mxstatus.IME` is hardwired to zero. On the other hand, there are nowhere to bypass instruction/data local memories even when `IME/DME` flags are set. So exception handlers should not be located in ILM or use data in DLM if recursive Parity/ECC exceptions are a concern.

The triggered exceptions could be either precise or imprecise. It all depends on whether the pipeline can attribute the errors to the faulting instruction and preserve the architectural states from being affected by the faulting instruction and its subsequent instructions. If these errors can be precise exceptions, access faults (precise exceptions) are triggered. Otherwise, local interrupts (imprecise exceptions) will be triggered. See the tables in Section 15.3.13 for how a trap handler can distinguish between them.

Information of the first detected precise Parity/ECC error will be logged into the associated ECC information registers (`mtval` and `mecc_code`). However, imprecise Parity/ECC errors are logged differently. When an imprecise Parity/ECC error exception is masked out and deferred, information from latter Parity/ECC errors will overwrite earlier ones. Additionally, only the `mecc_code` information of the last detected imprecise Parity/ECC error will be logged for imprecise exceptions.

When Parity/ECC errors occur on both the instruction fetch side and data access side of the processor sequentially, the first error will trigger the exception handler, and the second error may occur inside the exception handler, leading to re-entrance into the exception handler. This may corrupt `mepc`, making it impossible to resume application execution.

14.4 Error Handling in Caches

The behavior of the cache ECC logic is controlled by the following CSR. See Section 15.10.6 for more information.

- `mcache_ctl.IC_ECCEN`
- `mcache_ctl.DC_ECCEN`

The definitions of correctable and uncorrectable errors and their handling are summarized in Table 63 and Table 64.

Table 63: Handling of Correctable Errors in Caches

Error Type	Error Handling Action
One-bit parity errors or one-bit/two-bit ECC errors in clean cache lines	• Clean lines are invalidated and correct copies from the next-level memory are brought back into the cache. • All lines in I-Cache are considered clean.
One-bit ECC errors in dirty cache lines	• All data in dirty lines are written back to the next-level memory after ECC correction. • Dirty lines are then invalidated and correct copies from the next-level memory are brought back into the cache.

Table 64: Handling of Uncorrectable Errors in Caches

Error Type	Error Handling Action
One-bit parity errors or two-bit ECC errors in dirty cache lines	• All data in dirty lines are written back to the next-level memory. – Data without ECC errors and with one-bit ECC errors are written back after correction. – Data with one-bit parity error or two-bit ECC errors cannot be corrected and they are written back without correction. • The dirty lines are then invalidated.

14.5 Error Handling in ILM and DLM

The actions when a Parity/ECC error is detected in ILM/DLM are listed in Table 65.

Table 65: Local Memory Parity/ECC Error Handling

Error Type	Error Handling Action
Correctable errors	The data is corrected and written back to ILM/DLM.
Uncorrectable errors	The data is not corrected and an *Access Fault* is triggered with `mxstatus.DME` set.

14.6 Behavior of Local Memory Accesses Under Parity/ECC Configuration

The behavior of Local Memory ECC logic is controlled by `milmb.ECCEN/mdlmb.ECCEN`. See Section 15.10.1 and Section 15.10.2 for more information.

Table 66: Parity/ECC Behavior for Local Memory Operations

Operation	Parity/ECC Error Checking
Instruction fetches and load/store instructions	Controlled by `milmb.ECCEN/mdlmb.ECCEN`.
Slave port accesses	• Controlled by `milmb.ECCEN/mdlmb.ECCEN`. • No exception would be generated for detected Parity/ECC errors. • Uncorrectable errors would be reported through error response. • Reported errors controlled by `milmb.ECCEN/mdlmb.ECCEN` will trigger local interrupts.

Table 67: Types of Parity/ECC Error Exception

Access Type	Target RAM	Precise/Imprecise
Instruction Fetches	Local memories	Precise
Load-type instructions	Local memories	Precise
Store-type instructions	Local memories	Precise

15 Control and Status Registers

15.1 Introduction

The sections below describe the registers in detail.

15.1.1 System Register Type

Term	Description
IM	Implementation dependent/determined
IM Requirement	Conditions for registers to be present. “Required” means “always present”.
RO	Read-Only register/field. Any software write to RO register/field will be silently ignored by hardware.
RW	Read/Write register/field
W1	Write-only. Only writing 1 has an effect.
W1S	Write 1 to Set
W1C	Write 1 to Clear
WLRL	Write/Read Only Legal Values
WARL	Write-Any-Read-Legal

15.1.2 Reset Value

Term	Description
DC	The reset value is “Don’t Care”.

15.1.3 CSR Listing

Table 68: Machine Information Registers

Mnemonic Name	CSR Address	Definition
mvendorid	0xf11	Section 15.2.1
marchid	0xf12	Section 15.2.2
mimpid	0xf13	Section 15.2.3
mhartid	0xf14	Section 15.2.4

Table 69: Machine Trap Related Registers

Mnemonic Name	CSR Address	Definition
mstatus	0x300	Section 15.3.1
misa	0x301	Section 15.3.2
medeleg	0x302	Section 15.3.3
mideleg	0x303	Section 15.3.4
mie	0x304	Section 15.3.5
mtvec	0x305	Section 15.3.6
mscratch	0x340	Section 15.3.7
mepc	0x341	Section 15.3.8
mcause	0x342	Section 15.3.9
mtval	0x343	Section 15.3.10
mip	0x344	Section 15.3.11
mxstatus	0x7c4	Section 15.3.12
mdcause	0x7c9	Section 15.3.13

Table 70: Machine Counter Related Registers

Mnemonic Name	CSR Address	Definition
mcycle	0xb00	Section 15.4.1
minstret	0xb02	Section 15.4.2
mhpmcounter3	0xb03	Section 15.4.3
mhpmcounter4	0xb04	Section 15.4.3
mhpmcounter5	0xb05	Section 15.4.3
mhpmcounter6	0xb06	Section 15.4.3
mcounteren	0x306	Section 15.4.6
mhpmevent3	0x323	Section 15.4.5
mhpmevent4	0x324	Section 15.4.5
mhpmevent5	0x325	Section 15.4.5
mhpmevent6	0x326	Section 15.4.5
mcountinhibit	0x320	Section 15.4.4
mcounterwen	0x7ce	Section 15.4.7
mcounterinten	0x7cf	Section 15.4.8
mcountermask_m	0x7d1	Section 15.4.9
mcountermask_u	0x7d3	Section 15.4.10
mcounterovf	0x7d4	Section 15.4.11

Table 71: Configuration Control & Status Registers

Mnemonic Name	CSR Address	Definition
micm_cfg	0xfc0	Section 15.6.1
mdcm_cfg	0xfc1	Section 15.6.2
mmsc_cfg	0xfc2	Section 15.6.3
mrvarch_cfg	0xfca	Section 15.6.4

Table 72: Trigger Registers

Mnemonic Name	CSR Address	Definition
tselect	0x7a0	Section 15.7.1
tdata1	0x7a1	Section 15.7.2
tdata2	0x7a2	Section 15.7.3
tdata3	0x7a3	Section 15.7.4
tinfo	0x7a4	Section 15.7.5
tcontrol	0x7a5	Section 15.7.6
mcontext	0x7a8	Section 15.7.7
mcontrol	0x7a1	Section 15.7.8
icount	0x7a1	Section 15.7.9
itrigger	0x7a1	Section 15.7.10
etrigger	0x7a1	Section 15.7.11
textra	0x7a3	Section 15.7.12

Table 73: Debug Registers

Mnemonic Name	CSR Address	Definition
dcsr	0x7b0	Section 15.8.1
dpc	0x7b1	Section 15.8.2
dscratch0	0x7b2	Section 15.8.3
dscratch1	0x7b3	Section 15.8.4
dexc2dbg	0x7e0	Section 15.8.5
ddcause	0x7e1	Section 15.8.6

Table 74: User Trap Related Registers

Mnemonic Name	CSR Address	Definition
ustatus	0x000	Section 15.9.1
uie	0x004	Section 15.9.2
utvec	0x005	Section 15.9.3
uscratch	0x040	Section 15.9.4
uepc	0x041	Section 15.9.5
ucause	0x042	Section 15.9.6
utval	0x043	Section 15.9.7
uip	0x044	Section 15.9.8
udcause	0x809	Section 15.9.9

Table 75: User Counter Related Registers

Mnemonic Name	CSR Address	Definition
cycle	0xc00	Section 15.5.1
time	0xc01	Section 15.5.2
instret	0xc02	Section 15.5.3
hpmcounter3	0xc03	Section 15.5.4
hpmcounter4	0xc04	Section 15.5.4
hpmcounter5	0xc05	Section 15.5.4
hpmcounter6	0xc06	Section 15.5.4

Table 76: Memory and Miscellaneous Registers

Mnemonic Name	CSR Address	Definition
milmb	0x7c0	Section 15.10.1
mdlmb	0x7c1	Section 15.10.2
mecc_code	0x7c2	Section 15.10.3
mnvec	0x7c3	Section 15.10.4
mpft_ctl	0x7c5	Section 15.10.5
mcache_ctl	0x7ca	Section 15.10.6
mcctlbeginaddr	0x7cb	Section 15.10.8
mcctlcommand	0x7cc	Section 15.10.9
mcctldata	0x7cd	Section 15.10.10
ucctlbeginaddr	0x80b	Section 15.10.11

Continued on next page...

Table 76: (continued)

Mnemonic Name	CSR Address	Definition
ucctlcommand	0x80c	Section 15.10.12
mmisc_ctl	0x7d0	Section 15.10.7

Table 77: Hardware Stack Protection and Recording Registers

Mnemonic Name	CSR Address	Definition
mhsp_ctl	0x7c6	Section 15.11.1
msp_bound	0x7c7	Section 15.11.2
msp_base	0x7c8	Section 15.11.3

Table 78: CoDense Registers

Mnemonic Name	CSR Address	Definition
uitb	0x800	Section 15.12.1

Table 79: PMP Registers

Mnemonic Name	CSR Address	Definition
pmpcfg0, pmpcfg2	0x3a0, 0x3a2	Section 15.13.1
pmpaddr0–pmpaddr15	0x3b0–0x3bf	Section 15.13.2

15.2 Machine Information Registers

15.2.1 Machine Vendor ID Register

Mnemonic Name: mvendorid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf11 (standard read only)

63 … 0
MVENDORID

This read-only register provides the Andes JEDEC manufacturer ID: 0x0000031e.

Field Name	Bits	Description	Type	Reset
MVENDORID	[63:0]	The manufacturer ID of Andes	RO	0x0000031e

15.2.2 Machine Architecture ID Register

Mnemonic Name: marchid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf12 (standard read only)

63	62 … 31	30 … 0
1	0	CPU_ID

This register provides the micro-architecture id of AndesCore processor implementations. For NX25(F), `marchid.CPU_ID` will be 0x8025. Note that the MSB of this register is 1 for commercial implementations of RISC-V processors.

Field Name	Bits	Description	Type	Reset
CPU_ID	[30:0]	Andes CPU ID	RO	0x8025

15.2.3 Machine Implementation ID Register

Mnemonic Name: mimpid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf13 (standard read only)

63 — 32	31 — 8	7 — 4	3 — 0
0	MAJOR	MINOR	EXTENSION

This register is used to identify the revision number of the processor. Please see *AndesCore NX25(F) Release Note (RN169)* for exact values. It is documented in the release note as MAJOR.MINOR.EXTENSION.

Field Name	Bits	Description	Type	Reset
EXTENSION	[3:0]	Revision extension	RO	IM
MINOR	[7:4]	Revision minor	RO	IM
MAJOR	[31:8]	Revision major	RO	IM

15.2.4 Hart ID Register

Mnemonic Name: mhartid
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xf14 (standard read only)

63 … 0
MHARTID

This register provides the ID of the hardware thread. It is required that one of the hart IDs must be zero on a RISC-V platform. It is generally zero for single core systems,otherwise,consult your chip vendor for the reset value.

Field Name	Bits	Description	Type	Reset
MHARTID	[63:0]	Hart ID	RO	IM

15.3 Machine Trap Related CSRs

15.3.1 Machine Status

Mnemonic Name: mstatus
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x300 (standard read/write)

31–20	19	18	17	16–15	14–13	12–11	10–8	7	6–5	4	3	2–1	0
0	MXR	0	MPRV	XS	FS	MPP	0	MPIE	0	UPIE	MIE	0	UIE

63	62–36	35–34	33–32
SD	0	SXL	UXL

Field Name	Bits	Description	Type	Reset
UIE	[0]	U-mode interrupt enable bit. Value: 0 = Disabled; 1 = Enabled	RW	0

Value	Meaning
0	Disabled
1	Enabled

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MIE</td><td>[3]</td><td>M-mode interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UPIE</td><td>[4]</td><td>UPIE holds the value of the UIE bit prior to a trap.</td><td>RW</td><td>0</td></tr>
<tr><td>MPIE</td><td>[7]</td><td>MPIE holds the value of the MIE bit prior to a trap.</td><td>RW</td><td>0</td></tr>
<tr><td>MPP</td><td>[12:11]</td><td>MPP holds the privilege mode prior to a trap. Encoding for privilege mode is described in Table 4. When U-mode is not available, this field is hardwired to 3.</td><td>WARL</td><td>3</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
FS	[14:13]	`FS` holds the status of the architectural states of the floating-point unit, including the `fcsr` CSR and `f0` – `f31` floating-point data registers. The value of this field is zero and read-only if the processor does not have FPU. This field is primarily managed by software. The processor hardware assists the state managements in two regards: • Attempts to access `fcsr` or any `f` register raise an illegal-instruction exception when `FS` is Off. • `FS` is updated to the Dirty state with the execution of any instruction that updates `fcsr` or any `f` register when `FS` is Initial or Clean. Changing the setting of this field has no effect on the contents of the floating-point register states. In particular, setting `FS` to Off does not destroy the states, nor does setting `FS` to Initial clear the contents. **Value** / **Meaning**: 0 = Off; 1 = Initial; 2 = Clean; 3 = Dirty	WLRL	0

Value	Meaning
0	Off
1	Initial
2	Clean
3	Dirty

Continued on next page...

Field Name	Bits	Description	Type	Reset
XS	[16:15]	`XS` holds the status of the architectural states (ACE registers) of ACE instructions. The value of this field is zero if ACE extension is not configured. This field is primarily managed by software. The processor hardware assists the state managements in two regards: • Illegal instruction exceptions are triggered when `XS` is Off. • `XS` is updated to the Dirty state with the execution of ACE instructions when `XS` is not Off. Changing the setting of this field has no effect on the contents of ACE states. In particular, setting `XS` to Off does not destroy the states, nor does setting `XS` to Initial clear the contents. **Value** / **Meaning**: 0 – Off 1 – Initial 2 – Clean 3 – Dirty	RO	0
MPRV	[17]	When the `MPRV` bit is set, the memory access privilege for load and store are specified by the `MPP` field. When U-mode is not available, this field is hardwired to 0.	RW	0
MXR	[19]	`MXR` controls whether execute-only pages are readable. It has no effect when page-based virtual memory is not in effect. **Value** / **Meaning**: 0 – Execute-only pages are not readable 1 – Execute-only pages are readable	RW	0

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>UXL</td><td>[33:32]</td><td><code>UXL</code> controls the value of XLEN for U-mode. When U-mode is not available, this field is hardwired to 0.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>1</td><td>32</td></tr><tr><td>2</td><td>64</td></tr></table></td><td>RO</td><td>2/0</td></tr>
<tr><td>SXL</td><td>[35:34]</td><td><code>SXL</code> controls the value of XLEN for S-mode. When S-mode is not available, this field is hardwired to 0.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>1</td><td>32</td></tr><tr><td>2</td><td>64</td></tr></table></td><td>RO</td><td>2/0</td></tr>
<tr><td>SD</td><td>[63]</td><td><code>SD</code> summarizes whether either the <code>FS</code> field or <code>XS</code> field is dirty.</td><td>RO</td><td>0</td></tr>
</table>

When N extension is not supported, the corresponding bits in `mstatus` are hardwired to zero.

15.3.2 Machine ISA Register

Mnemonic Name: misa
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x301 (standard read/write)

63 62	61 26	25 0
Base	0	Extensions

Field Name	Bits	Description	Type	Reset
Extensions	[25:0]	See Table 80.	RO	IM

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Base</td><td>[63:62]</td><td>The general-purpose register width of the native base integer ISA.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Reserved</td></tr><tr><td>1</td><td>32</td></tr><tr><td>2</td><td>64</td></tr><tr><td>3</td><td>128</td></tr></table></td><td>RO</td><td>2</td></tr>
</table>

Table 80: RISC-V Definition of the Extensions Field

Bit	Extension	Description
0	A	Atomic extension
1	B	*Tentatively reserved for Bit operations extension*
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	Additional standard extensions present
7	H	Reserved
8	I	RV32I/64I/128I base ISA
9	J	*Tentatively reserved for Dynamically Translated Languages extension*
10	K	Reserved
11	L	*Tentatively reserved for Decimal Floating-Point extension*
12	M	Integer Multiply/Divide extension
13	N	User-level interrupts supported
14	O	Reserved
15	P	*Tentatively reserved for Packed-SIMD extension*
16	Q	Quad-precision floating-point extension
17	R	Reserved
18	S	Supervisor mode implemented
19	T	*Tentatively reserved for Transactional Memory extension*
20	U	User mode implemented
21	V	*Tentatively reserved for Vector extension*
22	W	Reserved

Continued on next page...

Table 80: (continued)

Bit	Extension	Description
23	X	Non-standard extensions present
24	Y	Reserved
25	Z	Reserved

15.3.3 Machine Exception Delegation

Mnemonic Name: medeleg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x302 (standard read/write)

63–16	15	14	13	12	11–10	9	8	7	6	5	4	3	2	1	0
0	SPF	0	LPF	IPF	0	SEC	UEC	SAF	SAM	LAF	LAM	B	II	IAF	IAM

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>IAM</td><td>[0]</td><td><code>IAM</code> indicates whether an Instruction Address Misaligned exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>IAF</td><td>[1]</td><td><code>IAF</code> indicates whether an Instruction Access Fault exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>II</td><td>[2]</td><td><code>II</code> indicates whether an Illegal Instruction exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>B</td><td>[3]</td><td>B indicates whether an exception triggered by breakpoint will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>LAM</td><td>[4]</td><td>LAM indicates whether a Load Address Misaligned exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>LAF</td><td>[5]</td><td>LAF indicates whether a Load Access Fault exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SAM</td><td>[6]</td><td>SAM indicates whether a Store/AMO Address Misaligned exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SAF</td><td>[7]</td><td>SAF indicates whether a Store/AMO Access Fault exception will be delegated to S-mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>UEC</td><td>[8]</td><td>UEC indicates whether an exception triggered by environment call from U-mode will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SEC</td><td>[9]</td><td>SEC indicates whether an exception triggered by environment call from S-mode will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>IPF</td><td>[12]</td><td>IPF indicates whether an Instruction Page Fault exception will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>LPF</td><td>[13]</td><td>LPF indicates whether a Load Page Fault exception will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SPF</td><td>[15]</td><td>SPF indicates whether a Store/AMO Page Fault exception will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

When supervisor mode or N extension is not supported, the corresponding bits in `medeleg` are hard-wired to zero.

15.3.4 Machine Interrupt Delegation

Mnemonic Name: mideleg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x303 (standard read/write)

63–10	9	8	7–6	5	4	3–2	1	0
0	SEI	UEI	0	STI	UTI	0	SSI	USI

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>USI</td><td>[0]</td><td><code>USI</code> indicates whether an U-mode software interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SSI</td><td>[1]</td><td><code>SSI</code> indicates whether an S-mode software interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UTI</td><td>[4]</td><td><code>UTI</code> indicates whether an U-mode timer interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>STI</td><td>[5]</td><td><code>STI</code> indicates whether an S-mode timer interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>UEI</td><td>[8]</td><td>UEI indicates whether an U-mode external interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SEI</td><td>[9]</td><td>SEI indicates whether an S-mode external interrupt will be delegated to S-mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not delegate</td></tr><tr><td>1</td><td>delegate</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

When supervisor mode or N extension is not supported, the corresponding bits in `mideleg` are hard-wired to zero.

15.3.5 Machine Interrupt Enable

Mnemonic Name: mie
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x304 (standard read/write)

63 19	18	17	16	15 12	11	10 9	8	7	6 5	4	3	2 1	0
0	PMOVI	BWEI	IMECCI	0	MEIE	0	UEIE	MTIE	0	UTIE	MSIE	0	USIE

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>USIE</td><td>[0]</td><td>U-mode software interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MSIE</td><td>[3]</td><td>M-mode software interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>UTIE</td><td>[4]</td><td>U-mode timer interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MTIE</td><td>[7]</td><td>M-mode timer interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UEIE</td><td>[8]</td><td>U-mode external interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MEIE</td><td>[11]</td><td>M-mode external interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>IMECCI</td><td>[16]</td><td>Imprecise ECC error local interrupt enable bit. The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>BWEI</td><td>[17]</td><td>Bus write transaction error local interrupt enable bit. The processor may receive bus errors on store instructions or cache writebacks.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>PMOVI</td><td>[18]</td><td>Performance monitor overflow local interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

When N extension is not supported, the corresponding bits in `mie` are hardwired to zero.

Each local interrupt can be configured with a local interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields above show the default bit location.

15.3.6 Machine Trap Vector Base Address

Mnemonic Name: mtvec
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x305 (standard read/write)

63 … 2	1 … 0
BASE[63:2]	0

This register determines the base address of the trap vector. The least significant 2 bits are hardwired to zeros. When the configured address width is less than 64, the upper bits are hardwired to zeros. When `mmisc_ctl.VEC_PLIC` is 0 (PLIC is not in the vector mode), this register indicates the entry points for the trap handler and it may point to any 4-byte aligned location in the memory space.

On the other hand, when `mmisc_ctl.VEC_PLIC` is 1 (PLIC is in the vector mode), this register will be the base address of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.

- This register should be aligned to $2^{\log_2 N+2}$-byte boundary for PLIC with N interrupt sources. For example, if N is 1023, the minimum alignment requirement is 4096 bytes (4 KiB).
- `mtvec[0]` is for exceptions and non-external local interrupts.
- `mtvec[`*i*`]` is for external PLIC interrupt source *i* triggered through the `mip.MEIP` pending condition.
- `mtvec[1024+`*i*`]` is for external PLIC interrupt source *i* triggered through
  - the `mip.UEIP` pending condition when `mideleg.UEI` == 0 for M/U systems.

Field Name	Bits	Description	Type	Reset
BASE[63:2]	[63:2]	Base address for interrupt and exception handlers. See description above for alignment requirements when PLIC is in the vector mode.	RW	0

15.3.7 Machine Scratch Register

Mnemonic Name: mscratch
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x340 (standard read/write)

63 … 0
MSCRATCH

This is a scratch register for temporary data storage, which is typically used by the M-mode trap handler.

Field Name	Bits	Description	Type	Reset
MSCRATCH	[63:0]	Scratch register storage.	RW	0

15.3.8 Machine Exception Program Counter

Mnemonic Name: mepc
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x341 (standard read/write)

63 … 1	0
EPC	0

This register is written with the virtual address of the instruction that encountered traps and/or NMIs when these events occurred.

Field Name	Bits	Description	Type	Reset
EPC	[63:1]	Exception program counter.	RW	0

15.3.9 Machine Cause Register

Mnemonic Name: mcause
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x342 (standard read/write)

63	62 … 12	11 … 0
INTERRUPT	0	EXCEPTION_CODE

This register indicates the cause of trap, reset, NMI or the interrupt source ID of a vector interrupt. This register is updated when a trap, reset, NMI or vector interrupt occurs. Please see Section 11 for an overview of how interrupts and exceptions are handled by the processor. When multiple events may cause a trap to be taken with the same `mcause` value, the value of mdcause records the exact event that causes the trap.

Exceptions can be precise or imprecise. Only precise exceptions are triggered as the standard RISC-V exceptions with the `mcause.INTERRUPT` bit clear. Imprecise exceptions are triggered as local interrupts, with the `mcause.INTERRUPT` bit set.

Field Name	Bits	Description	Type	Reset
EXCEPTION_CODE	[11:0]	Exception code	RW	0
INTERRUPT	[63]	Interrupt	RW	0

Note

For CPU revisions equal to or earlier than 1.4.0, width of `EXCEPTION_CODE` is defined to be 6-bit wide ([5:0]). It is extended to 12 bits ([11:0]) to support local interrupt sources up to 4096 sources.

The following tables show the possible values of `mcause`:

Table 81: Possible Values of mcause After Trap

Interrupt	Exception Code	Description
1	0	User software interrupt
1	3	Machine software interrupt
1	4	User timer interrupt
1	7	Machine timer interrupt
1	8	User external interrupt
1	11	Machine external interrupt

Continued on next page. . .

Table 81: (continued)

Interrupt	Exception Code	Description
1	16	Imprecise ECC error interrupt (slave port accesses and D-Cache evictions) (M-mode)
1	17	Bus read/write transaction error interrupt (M-mode)
1	18	Performance monitor overflow interrupt (M-mode)
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/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	11	Environment call from M-mode
0	32	Stack overflow exception
0	33	Stack underflow exception
0	40–47	Andes Custom Extension exception (see *Andes Custom Extension Specification* for more details)

Table 82: Possible Values of mcause After Reset

Interrupt	Exception Code	Description
0	0	Initial value when the processor comes out of reset (by `core_reset_n`)

Table 83: Possible Values of mcause After NMI

Interrupt	Exception Code	Description
0	0x001	NMI triggered

Table 84: Possible Values of mcause After Vector Interrupt

mcause	Description
Interrupt source ID	Interrupt source ID when a vector interrupt occurs

15.3.10 Machine Trap Value

Mnemonic Name: mtval
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x343 (standard read/write)

63 ... 0
MTVAL

This register is updated when a trap is taken to M-mode. The updated value is dependent on the cause of traps:

- For Hardware Breakpoint exceptions, Address Misaligned exceptions, or Access Fault exceptions, it is the effective faulting addresses.
- For illegal instruction exceptions, the updated value is the faulting instruction. If the length of the instruction is less than XLEN bits long, the upper bits of `mtval` are cleared to zero. For the EXEC.IT instructions triggering illegal instruction exceptions, the faulting instruction is the translated instruction. Please note that if an EXEC.IT instruction is translated to a 16-bit instruction, the translated instruction is considered an illegal instruction even if it is normally a valid one.
- For other exceptions, `mtval` is set to zero.

For instruction-fetch access faults, this register will be updated with the address pointing to the portion of the instruction that caused the fault, while the `mepc` register will be updated with the address pointing to the beginning of the instruction.

When the configured address width is less than 64, the upper bits of `mtval` are hardwired to zeros.

Field Name	Bits	Description	Type	Reset
MTVAL	[63:0]	Exception-specific information for software trap handling.	RW	0

15.3.11 Machine Interrupt Pending

Mnemonic Name: mip
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x344 (standard read/write)

63 19	18	17	16	15 12	11	10 9	8	7	6 5	4	3	2 1	0
0	PMOVI	BWEI	IMECCI	0	MEIP	0	UEIP	MTIP	0	UTIP	MSIP	0	USIP

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>USIP</td><td>[0]</td><td>U-mode software interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MSIP</td><td>[3]</td><td>M-mode software interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>UTIP</td><td>[4]</td><td>U-mode timer interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MTIP</td><td>[7]</td><td>M-mode timer interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>UEIP</td><td>[8]</td><td>U-mode external interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MEIP</td><td>[11]</td><td>M-mode external interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>IMECCI</td><td>[16]</td><td>Imprecise ECC error local interrupt pending bit. The processor may receive imprecise ECC errors on slave port accesses or cache writebacks.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>BWEI</td><td>[17]</td><td>Bus write transaction error local interrupt pending bit. The processor may receive bus errors on store instructions or cache writebacks.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>PMOVI</td><td>[18]</td><td>Performance monitor overflow local interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

When N extension is not supported, the corresponding bits in `mip` are hardwired to zero.

Each local interrupt can be configured with a local interrupt number. Bit location of interrupts are the same as their interrupt numbers. Register fields above show the default bit location.

15.3.12 Machine Extended Status

Mnemonic Name: mxstatus
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7c4 (non-standard read/write)

63 … 10	9 … 6	5	4	3	2	1	0
0	0	PDME	DME	PIME	IME	PPFT_EN	PFT_EN

Field Name	Bits	Description	Type	Reset
PFT_EN	[0]	Enable performance throttling. When throttling is enabled, the processor executes instructions at the performance level specified in `mpft_ctl.T_LEVEL`. On entering a trap: • `PPFT_EN` ⇐ `PFT_EN`; • `PFT_EN` ⇐ `mpft_ctl.FAST_INT` ? 0 : `PFT_EN`; On executing an `MRET` instruction: • `PFT_EN` ⇐ `PPFT_EN`; This field is hardwired to 0 if the PowerBrake feature is not supported.	RW	0
PPFT_EN	[1]	For saving previous `PFT_EN` state on entering a trap. This field is hardwired to 0 if the PowerBrake feature is not supported.	RW	0
IME	[2]	Instruction Machine Error flag. It indicates an exception occurred at the instruction cache or instruction local memory (ILM) and the respective memory should be bypassed. Because of the following reasons, this field is hardwired to 0: • As all I-Cache ECC errors can be repaired and there is no need to bypass it • There is no memory behind ILM for bypassing.	RO	0
PIME	[3]	For saving previous `IME` state on entering a trap. This field is hardwired to 0.	RO	0
DME	[4]	Data Machine Error flag. It indicates an exception occurred at the data cache or data local memory (DLM). Load/store accesses will bypass D-Cache when this bit is set. The exception handler should clear this bit after the machine error has been dealt with. It will be set by Data parity/ECC error exceptions.	RW	0

Continued on next page...

Field Name	Bits	Description	Type	Reset
PDME	[5]	For saving previous DME state on entering a trap. This field is hardwired to 0 if data cache and data local memory are not supported.	RW	0

15.3.13 Machine Detailed Trap Cause

Mnemonic Name: mdcause
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7c9 (non-standard read/write)

63 … 3	2 … 0
0	MDCAUSE

Field Name	Bits	Description	Type	Reset
MDCAUSE	[2:0]	This register further disambiguates causes of traps recorded in the `mcause` register. See the list below for details.	RW	0

The value of `MDCAUSE` for precise exception:

- When `mcause` == 1 (Instruction access fault):

Value	Meaning
0	Reserved
1	ECC/Parity error
2	PMP instruction access violation
3	Bus error
4	Reserved

- When `mcause` == 2 (Illegal instruction):

Value	Meaning
0	The actual faulting instruction is stored in the `mtval` CSR.

Continued on next page...

Value	Meaning
1	FP disabled exception
2	ACE disabled exception

- When `mcause` == 5 (Load access fault)

Value	Meaning
0	Reserved
1	ECC/Parity error
2	PMP load access violation
3	Bus error
4	Misaligned address
5	Reserved
6	PMA attribute inconsistency
7	Reserved

- When `mcause` == 7 (Store access fault)

Value	Meaning
0	Reserved
1	ECC/Parity error
2	PMP store access violation
3	Bus error
4	Misaligned address
5	Reserved
6	PMA attribute inconsistency
7	Reserved

- For other exceptions and interrupts, this register will not be updated.

15.3.13.1 Detailed Exception Priority

Within Instruction/Load/Store access fault exceptions, the priority of a PMP exception is higher than the priority of a PMA exception, when both types of exceptions happen on the same instruction.

15.4 Machine Counter Related CSRs

15.4.1 Machine Cycle Counter

Mnemonic Name: mcycle
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb00 (standard read/write)

The `mcycle` CSR counts the number of cycles that the hart has executed since some arbitrary time in the past. The `mcycle` register has 64-bit precision.

15.4.2 Machine Instruction-Retired Counter

Mnemonic Name: minstret
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb02 (standard read/write)

The `minstret` CSR counts the number of instructions that the hart has retired since some arbitrary time in the past. The `minstret` register has 64-bit precision.

15.4.3 Machine Performance Monitoring Counter

Mnemonic Name: mhpmcounter3–mhpmcounter6
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xb03 to 0xb06 (standard read/write)

The `mhpmcounter3–mhpmcounter6` CSRs count the number of events selected by `mhpmevent3–mhpmevent6`.

15.4.4 Machine Counter-Inhibit

Mnemonic Name: mcountinhibit
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x320 (non-standard read/write)

63 – 7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The counter-inhibit register controls which counters should not be incremented. When the `CY`, `IR`, or `HPMn` bit is set, the corresponding counter will not be incremented on the event.

15.4.5 Machine Performance Monitoring Event Selector

Mnemonic Name: mhpmevent3–mhpmevent6
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x323 to 0x326 (standard read/write)

63 – 9	8 – 4	3 – 0
0	SEL	TYPE

The event selectors are defined in Table 85. Micro-architectural events are mostly speculative in nature. The counted events include events caused by speculative actions, unless they are defined to be non-speculative in the comment section. In particular, *retired* instruction counts are non-speculative.

Table 85: Event Selectors

TYPE	SEL	Event Name	Comment
0	0	No event	Disable this counter.
0	1	Cycle count	Number of elapsed processor clock cycles.
0	2	Retired instruction count	Number of retired instructions.
0	3	Integer load instruction count	Number of retired load instructions (including `LR`).
0	4	Integer store instruction count	Number of retired store instructions (including `SC`).
0	5	Atomic instruction count	Number of retired atomic instructions (not including `LR` and `SC`).
0	6	System instruction count	Number of retired SYSTEM instructions (instructions with major opcode equal to 0b1110011).
0	7	Integer computational instruction count	Number of retired integer computational instructions.
0	8	Conditional branch instruction count	Number of retired conditional branch instructions.
0	9	Taken conditional branch instruction count	Number of retired conditional branch instructions that are taken.
0	10	`JAL` instruction count	Number of retired `JAL` instructions.
0	11	`JALR` instruction count	Number of retired `JALR` instructions. This event selector also counts the events monitored by the *return instruction count* event selector defined in the next row.
0	12	Return instruction count	Number of retired return instructions. Return instructions are `JALR` instructions with zero immediate offset and the following operands: • (rd != x1/x5) and (rs1 == x1/x5) • rd == x1 and rs1 == x5 • rd == x5 and rs1 == x1
0	13	Control transfer instruction count	Number of retired unconditional jumps (`JAL` and `JALR`) and conditional branch instructions.
0	14	`EXEC.IT` instruction count	Number of retired `EXEC.IT` instructions.

Continued on next page...

Table 85: (continued)

TYPE	SEL	Event Name	Comment
0	15	Integer multiplication instruction count	Number of retired integer multiplication instructions.
0	16	Integer division instruction count	Number of retired integer division/remainder instructions.
0	17	Floating-point load instruction count	Number of retired floating-point load instructions.
0	18	Floating-point store instruction count	Number of retired floating-point store instructions.
0	19	Floating-point addition instruction count	Number of retired floating-point addition/subtraction instructions.
0	20	Floating-point multiplication instruction count	Number of retired floating-point multiplication instructions.
0	21	Floating-point fused multiply-add instruction count	Number of retired floating-point fused multiply-add/subtraction instructions (`FMADD`, `FMSUB`, `FNMSUB`, `FNMADD`).
0	22	Floating-point division or square-root instruction count	Number of retired floating-point division/square-root instructions.
0	23	Other floating-point instruction count	Number of retired floating-point instructions not counted by the previous floating-point instruction event selectors.
1	0	ILM access	Number of ILM transfers, including speculative instruction fetch, load/store accesses, ECC repair and slave port accesses.
1	1	DLM access	Number of DLM transfers, including speculative load/store accesses, ECC repair and slave port accesses.
1	2	I-Cache access	Number of completed I-Cache fetch access.
1	3	I-Cache miss	Number of I-Cache fetch miss.
1	4	D-Cache access*	Number of completed D-Cache load-and-store access. Misaligned load/store accesses might increase this counter by either one or two, depending on access sizes and alignments. Only misaligned accesses crossing two cache lines are guaranteed to result in increment of two.

Continued on next page...

Table 85: (continued)

TYPE	SEL	Event Name	Comment
1	5	D-Cache miss*	The event counts the number of D-Cache load-and-store miss. Misaligned load/store accesses might increase this counter by either zero, one or two, depending on access sizes, alignments and whether the accessed lines are in D-Cache.
1	6	D-Cache load access*	Number of completed D-Cache load access. See the D-Cache access count event selector for the handling of misaligned load accesses.
1	7	D-Cache load miss*	Number of D-Cache load miss. See the D-Cache miss count event selector for the handling of misaligned load accesses.
1	8	D-Cache store access*	Number of completed D-Cache store access. See the D-Cache access count event selector for the handling of misaligned load accesses.
1	9	D-Cache store miss*	Number of D-Cache store miss. See the D-Cache miss count event selector for the handling of misaligned load accesses.
1	10	D-Cache writeback*	Number of D-Cache writeback.
1	11	Cycles waiting for I-Cache fill data*	Number of cycles waiting for the return of the critical word of I-Cache misses from the system bus. This event selector does not monitor accesses to I/O regions or accesses to cacheable regions when I-Cache is turned off.
1	12	Cycles waiting for D-Cache fill data*	Number of cycles waiting for the return of the critical word of D-Cache misses from the system bus. This event selector does not monitor accesses to I/O regions or accesses to cacheable regions when D-Cache is turned off.
1	13	Uncached fetch data access from bus*	Number of accesses for the instruction data to return from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when I-Cache is not configured or off.

Continued on next page. . .

Table 85: (continued)

TYPE	SEL	Event Name	Comment
1	14	Uncached load data access from bus*	Number of accesses for the load data to return from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when D-Cache is not configured or off.
1	15	Cycles waiting for uncached fetch data from bus*	Number of cycles waiting for uncached instruction data returning from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when I-Cache is not configured or off.
1	16	Cycles waiting for uncached load data from bus*	Number of cycles waiting for uncached load data returning from the system bus. This event selector monitors accesses to I/O regions or accesses to cacheable regions when D-Cache is not configured or off.
2	0	Misprediction of conditional branches (direction)	Number of misprediction of committed conditional branches.
2	1	Misprediction of taken conditional branches (direction)	Number of misprediction of committed taken conditional branches.
2	2	Misprediction of targets of Return instructions	Number of misprediction of committed Return instruction.
2	3	Replay for load-after-store or store-after-store cases	A load-after-store replay happens when a load hits a prior store in the pipeline with overlapping addresses. A store-after-store replay happens when a store hits a prior store in the pipeline with overlapping addresses.

Note

- Interrupts are expected to be disabled when monitoring D-Cache related events and cycles waiting related events.

15.4.6 Machine Counter Enable

Mnemonic Name: mcounteren
IM Requirement: Required if User mode is implemented
Access Mode: Machine
CSR Address: 0x306 (standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	TM	CY

The machine counter-enable register controls the availability of the hardware performance monitoring counters to the next-lowest privileged mode. The default value of this register is 0.

When `CY`, `TM`, `IR`, `HPM3`, `HPM4`, `HPM5`, or `HPM6` in the `mcounteren` register is 0, attempts to read the `cycle`, `time`, `instret`, `hpmcounter3`, `hpmcounter4`, `hpmcounter5`, or `hpmcounter6` registers while executing in U-mode (M/U configuration) will cause an illegal instruction exception. When one of these bits is set, accessing to the corresponding register is permitted in the next implemented privilege mode.

15.4.7 Machine Counter Write Enable

Mnemonic Name: mcounterwen
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7CE (non-standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The machine counter write enable register controls the permission of writing the hardware performance monitoring counters in the next-lowest privileged mode and M-mode itself. The default value of this register is 0.

When `CY`, `IR`, `HPM3`, `HPM4`, `HPM5`, or `HPM6` in the `mcounterwen` register is 0, attempts to write the `cycle`, `time`, `instret`, `hpmcounter3`, `hpmcounter4`, `hpmcounter5`, or `hpmcounter6` registers while executing in U-mode or M-mode (M/U configuration) will cause an illegal instruction exception. When one of these bits is set, writing to the corresponding register is permitted in M-mode and the next implemented privilege mode.

15.4.8 Machine Counter Interrupt Enable

Mnemonic Name: mcounterinten
IM Requirement: mmsc_cfg.PMNDS == 1
Access Mode: Machine
CSR Address: 0x7CF (non-standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The machine counter interrupt enable register controls whether a counter overflow interrupt is generated or not. The default value of this register is 0.

When `CY`, `IR`, `HPM3`, `HPM4`, `HPM5`, or `HPM6` in the `mcounterinten` register is 0, no overflow interrupt is generated for the corresponding counter. When one of these bits is set, an interrupt will be generated when the corresponding counter overflows (the counter value wraps around back to 0).

15.4.9 Machine Counter Mask for Machine Mode

Mnemonic Name: mcountermask_m
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7D1 (non-standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The machine counter mask for M-mode register controls the performance counter behavior in M-mode. The default value of this register is 0.

When `CY`, `IR`, `HPM3`, `HPM4`, `HPM5`, or `HPM6` in the `mcountermask_m` register is set, the specific counter will not be incremented in M-mode.

The setting in this register also controls the privileged mode of the overflow local interrupt when the corresponding counter overflows for the M /U configuration: For any bit in this register, overflow of the corresponding counter will trigger an M-mode interrupt if the bit is zero .

On the other hand, a counter overflow will always generate an M-mode interrupt for the M/U configuration, regardless of the settings in this register.

15.4.10 Machine Counter Mask for User Mode

Mnemonic Name: mcountermask_u
IM Requirement: mmsc_cfg.PMNDS == 1 and misa[20] == 1
Access Mode: Machine
CSR Address: 0x7D3 (non-standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The machine counter mask for U-mode register controls the performance counter behavior in U-mode. The default value of this register is 0.

15.4.11 Machine Counter Overflow Status

Mnemonic Name: mcounterovf
IM Requirement: mmsc_cfg.PMNDS == 1
Access Mode: Machine
CSR Address: 0x7D4 (non-standard read/write)

63–7	6	5	4	3	2	1	0
0	HPM6	HPM5	HPM4	HPM3	IR	0	CY

The machine counter overflow status register records the overflow status of performance counters. When a bit is set, it indicates that an overflow has happened to the corresponding counter. Write 1 to each bit will clear the overflow state for the corresponding counter.*

Note

- Under the write-1-clear scheme, the behavior of CSRRS and CSRRC is undefined. Software should use CSRRW to clear the overflow state.

15.5 User Counter Related CSRs

15.5.1 Cycle Counter

Mnemonic Name: cycle
IM Requirement: misa.U == 1
Access Mode: User
CSR Address: 0xc00 (standard read/write)

This register is the read-only shadow of `mcycle` register. Writing of this register in any mode will cause an illegal instruction exception. When the `mcounteren.CY` bit is cleared, attempts to read this register in User mode will also cause an illegal instruction exception.

When Andes Enhanced Performance Monitoring is configured (`mmsc_cfg.PMNDS=1`), setting the corresponding bit in `mcounterwen` allows CSR writes to this register in Machine and User mode for an M/U system. Otherwise, writing of this register will cause an illegal instruction exception.

15.5.2 User Time Register

Mnemonic Name: time
IM Requirement: misa.U==1
Access Mode: User
CSR Address: 0xc01 (software emulation)

This is the register for RDTIME instruction. It is not implemented by NX25(F) and an illegal instruction exception would be raised for accessing this register. The machine mode trap handler should get the timer value by loading mtime from the Machine Timer to emulate the correct behavior of a RDTIME instruction.

15.5.3 Instruction-Retired Counter

Mnemonic Name: instret
IM Requirement: misa.U == 1
Access Mode: User
CSR Address: 0xc02 (standard read/write)

This register is the read-only shadow of `minstret` register. Writing of this register in any mode will cause an illegal instruction exception. When the `mcounteren.IR` bit is cleared, attempts to read this register in User mode will cause an illegal instruction exception.

When Andes Enhanced Performance Monitoring is configured (`mmsc_cfg.PMNDS=1`), setting the corresponding bit in `mcounterwen` allows CSR writes to this register in Machine and User mode for an M/U system. Otherwise, writing of this register will cause an illegal instruction exception.

15.5.4 Performance Monitoring Counter

Mnemonic Name: hpmcounter3–hpmcounter6
IM Requirement: misa.U == 1
Access Mode: User
CSR Address: 0xc03 to 0xc06 (standard read/write)

These registers are the read-only shadow of `mhpmcounter3-mhpmcounter6` registers. Writing of these registers in any mode will cause an illegal instruction exception. When the `mcounteren.HPM3-6` bits are cleared, attempts to read these registers in User mode will cause an illegal instruction exception.

When Andes Enhanced Performance Monitoring is configured (`mmsc_cfg.PMNDS=1`), setting the corresponding bit in `mcounterwen` allows CSR writes to this register in Machine and User mode for an M/U system. Otherwise, writing of this register will cause an illegal instruction exception.

15.6 Configuration Control & Status Registers

15.6.1 Instruction Cache/Memory Configuration Register

Mnemonic Name: micm_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc0 (non-standard read only)

63–27	26–25	24	23	22–21	20	19–15	14–12	11–10	9	8–6	5–3	2–0
0	0	SETH	0	ILM_ECC	0	ILMSZ	ILMB	IC_ECC	ILCK	ISZ	IWAY	ISET

This register provides information about configurations of instruction cache and instruction memory.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ISET</td><td>[2:0]</td><td>I-Cache sets (# of cache lines per way):
When <code>micm_cfg.SETH==0</code>:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>64</td></tr>
<tr><td>1</td><td>128</td></tr>
<tr><td>2</td><td>256</td></tr>
<tr><td>3</td><td>512</td></tr>
<tr><td>4</td><td>1024</td></tr>
<tr><td>5</td><td>2048</td></tr>
<tr><td>6</td><td>4096</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
</table>
When <code>micm_cfg.SETH==1</code>:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>32</td></tr>
<tr><td>1</td><td>16</td></tr>
<tr><td>2</td><td>8</td></tr>
<tr><td>3~7</td><td>Reserved</td></tr>
</table>
• When instruction cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>IWAY</td><td>[5:3]</td><td>Associativity of I-Cache
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Direct-mapped</td></tr>
<tr><td>1</td><td>2-way</td></tr>
<tr><td>2</td><td>3-way</td></tr>
<tr><td>3</td><td>4-way</td></tr>
<tr><td>4</td><td>5-way</td></tr>
<tr><td>5</td><td>6-way</td></tr>
<tr><td>6</td><td>7-way</td></tr>
<tr><td>7</td><td>8-way</td></tr>
</table>
• When instruction cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ISZ</td><td>[8:6]</td><td>I-Cache block (line) size
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No I-Cache</td></tr><tr><td>1</td><td>8 bytes</td></tr><tr><td>2</td><td>16 bytes</td></tr><tr><td>3</td><td>32 bytes</td></tr><tr><td>4</td><td>64 bytes</td></tr><tr><td>5</td><td>128 bytes</td></tr><tr><td>6,7</td><td>Reserved</td></tr></table>• When instruction cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>ILCK</td><td>[9]</td><td>I-Cache locking support
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No locking support</td></tr><tr><td>1</td><td>With locking support</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>IC_ECC</td><td>[11:10]</td><td>I-Cache soft-error protection scheme
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No parity/ECC</td></tr><tr><td>1</td><td>Parity</td></tr><tr><td>2</td><td>ECC</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>ILMB</td><td>[14:12]</td><td>Number of ILM base registers present
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No ILM base register present</td></tr><tr><td>1</td><td>One ILM base register present</td></tr><tr><td>2-7</td><td>Reserved</td></tr></table>• When ILM is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ILMSZ</td><td>[19:15]</td><td>ILM Size
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>0 Byte</td></tr>
<tr><td>1</td><td>1 KiB</td></tr>
<tr><td>2</td><td>2 KiB</td></tr>
<tr><td>3</td><td>4 KiB</td></tr>
<tr><td>4</td><td>8 KiB</td></tr>
<tr><td>5</td><td>16 KiB</td></tr>
<tr><td>6</td><td>32 KiB</td></tr>
<tr><td>7</td><td>64 KiB</td></tr>
<tr><td>8</td><td>128 KiB</td></tr>
<tr><td>9</td><td>256 KiB</td></tr>
<tr><td>10</td><td>512 KiB</td></tr>
<tr><td>11</td><td>1 MiB</td></tr>
<tr><td>12</td><td>2 MiB</td></tr>
<tr><td>13</td><td>4 MiB</td></tr>
<tr><td>14</td><td>8 MiB</td></tr>
<tr><td>15</td><td>16 MiB</td></tr>
<tr><td>16-31</td><td>Reserved</td></tr>
</table>
• When ILM is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>ILM_ECC</td><td>[22:21]</td><td>ILM soft-error protection scheme
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No parity/ECC</td></tr>
<tr><td>1</td><td>Parity</td></tr>
<tr><td>2</td><td>ECC</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>SETH</td><td>[24]</td><td>This bit extends the `ISET` field.
• When instruction cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
</table>

15.6.2 Data Cache/Memory Configuration Register

Mnemonic Name: mdcm_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc1 (non-standard read only)

63–27	26–25	24	23	22–21	20	19–15	14–12	11–10	9	8–6	5–3	2–0
0	0	SETH	0	DLM_ECC	0	DLMSZ	DLMB	DC_ECC	DLCK	DSZ	DWAY	DSET

This register provides information about the configurations of data cache and data local memory.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DSET</td><td>[2:0]</td><td>D-Cache sets (# of cache lines per way):
When <code>mdcm_cfg.SETH</code>==0:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>64</td></tr>
<tr><td>1</td><td>128</td></tr>
<tr><td>2</td><td>256</td></tr>
<tr><td>3</td><td>512</td></tr>
<tr><td>4</td><td>1024</td></tr>
<tr><td>5</td><td>2048</td></tr>
<tr><td>6</td><td>4096</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
</table>
When <code>mdcm_cfg.SETH</code>==1:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>32</td></tr>
<tr><td>1</td><td>16</td></tr>
<tr><td>2</td><td>8</td></tr>
<tr><td>3~7</td><td>Reserved</td></tr>
</table>
• When data cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DWAY</td><td>[5:3]</td><td>Associativity of D-Cache
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Direct-mapped</td></tr>
<tr><td>1</td><td>2-way</td></tr>
<tr><td>2</td><td>3-way</td></tr>
<tr><td>3</td><td>4-way</td></tr>
<tr><td>4</td><td>5-way</td></tr>
<tr><td>5</td><td>6-way</td></tr>
<tr><td>6</td><td>7-way</td></tr>
<tr><td>7</td><td>8-way</td></tr>
</table>
• When data cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>DSZ</td><td>[8:6]</td><td>D-Cache block (line) size
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No D-Cache</td></tr>
<tr><td>1</td><td>8 bytes</td></tr>
<tr><td>2</td><td>16 bytes</td></tr>
<tr><td>3</td><td>32 bytes</td></tr>
<tr><td>4</td><td>64 bytes</td></tr>
<tr><td>5</td><td>128 bytes</td></tr>
<tr><td>6,7</td><td>Reserved</td></tr>
</table>
• When data cache is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>DLCK</td><td>[9]</td><td>D-Cache locking support
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No locking support</td></tr>
<tr><td>1</td><td>With locking support</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>DC_ECC</td><td>[11:10]</td><td>D-Cache soft-error protection scheme
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No parity/ECC support</td></tr>
<tr><td>1</td><td>Has parity support</td></tr>
<tr><td>2</td><td>Has ECC support</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DLMB</td><td>[14:12]</td><td>Number of DLM base registers present
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No DLM base register present</td></tr>
<tr><td>1</td><td>One DLM base register present</td></tr>
<tr><td>2-7</td><td>Reserved</td></tr>
</table>
• When DLM is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>DLMSZ</td><td>[19:15]</td><td>DLM Size
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>0 Byte</td></tr>
<tr><td>1</td><td>1 KiB</td></tr>
<tr><td>2</td><td>2 KiB</td></tr>
<tr><td>3</td><td>4 KiB</td></tr>
<tr><td>4</td><td>8 KiB</td></tr>
<tr><td>5</td><td>16 KiB</td></tr>
<tr><td>6</td><td>32 KiB</td></tr>
<tr><td>7</td><td>64 KiB</td></tr>
<tr><td>8</td><td>128 KiB</td></tr>
<tr><td>9</td><td>256 KiB</td></tr>
<tr><td>10</td><td>512 KiB</td></tr>
<tr><td>11</td><td>1 MiB</td></tr>
<tr><td>12</td><td>2 MiB</td></tr>
<tr><td>13</td><td>4 MiB</td></tr>
<tr><td>14</td><td>8 MiB</td></tr>
<tr><td>15</td><td>16 MiB</td></tr>
<tr><td>16-31</td><td>Reserved</td></tr>
</table>
• When DLM is not configured, this field should be ignored.</td><td>RO</td><td>IM</td></tr>
<tr><td>DLM_ECC</td><td>[22:21]</td><td>DLM soft-error protection scheme
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No parity/ECC</td></tr>
<tr><td>1</td><td>Parity</td></tr>
<tr><td>2</td><td>ECC</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
SETH	[24]	This bit extends the `DSET` field. • When data cache is not configured, this field should be ignored.	RO	IM

15.6.3 Misc. Configuration Register

Mnemonic Name: mmsc_cfg
IM Requirement: Required
Access Mode: Machine
CSR Address: 0xfc2 (non-standard read only)

14	13	12	11–7	6	5	4	3	2–1	0
LMSLVP	EV5PE	VPLIC	0	ACE	HSP	PFT	ECD	TLB_ECC	ECC

31	30	29	28–20	19–18	17	16	15
0	0	0	0	VCCTL	EFHW	CCTLCSR	PMNDS

63–53	52	51–45	44	43	42–41	40	39–36	35	34	33	32
0	RVARCH	0	0	VDOT	0	0	0	0	0	0	0

This register provides information regarding miscellaneous processor configurations.

Field Name	Bits	Description	Type	Reset
ECC	[0]	Indicates whether the parity/ECC soft-error protection is implemented or not. Value 0: Not implemented. Value 1: Implemented. The specific parity/ ECC scheme used for each protected RAM is specified by the control bits in the following list. • `micm_cfg.IC_ECC` • `micm_cfg.ILM_ECC` • `mdcm_cfg.DC_ECC` • `mdcm_cfg.DLM_ECC` • `mmsc_cfg.TLB_ECC`	RO	IM

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>TLB_ECC</td><td>[2:1]</td><td>TLB parity/ECC support configuration.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No parity/ECC support.</td></tr>
<tr><td>1</td><td>Has parity support.</td></tr>
<tr><td>2</td><td>Has ECC support.</td></tr>
<tr><td>3</td><td>Reserved.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>ECD</td><td>[3]</td><td>Indicates whether the Andes CoDense is implemented or not. This field is hardwired to 1.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Not implemented.</td></tr>
<tr><td>1</td><td>Implemented.</td></tr>
</table></td><td>RO</td><td>1</td></tr>
<tr><td>PFT</td><td>[4]</td><td>Indicates whether the Andes PowerBrake (Performance Throttling) power/performance scaling extension is implemented or not.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Not implemented.</td></tr>
<tr><td>1</td><td>Implemented.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>HSP</td><td>[5]</td><td>Indicates whether the Andes StackSafe hardware stack protection extension is implemented or not.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Not implemented.</td></tr>
<tr><td>1</td><td>Implemented.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>ACE</td><td>[6]</td><td>Indicates whether Andes Custom Extension is implemented or not.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Not implemented.</td></tr>
<tr><td>1</td><td>Implemented.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>VPLIC</td><td>[12]</td><td>Indicates whether the Andes Vectored PLIC Extension is implemented or not.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Not implemented.</td></tr>
<tr><td>1</td><td>Implemented.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
<tr><td>EV5PE</td><td>[13]</td><td>Indicates whether AndeStar V5 Performance Extension is implemented or not. The processor always implements AndeStar V5 Performance Extension.</td><td>RO</td><td>1</td></tr>
<tr><td>LMSLVP</td><td>[14]</td><td>Indicates if local memory slave port is present or not.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Local memory slave port is not present.</td></tr>
<tr><td>1</td><td>Local memory slave port is implemented.</td></tr>
</table>
Note that atomicity of atomic instructions accessing local memory address space is not guaranteed if external masters modify the same data through the local memory slave port.</td><td>RO</td><td>IM</td></tr>
<tr><td>PMNDS</td><td>[15]</td><td>Indicates if Andes performance monitoring feature is present or not. This feature can be selected by “Performance Monitors” in the configuration tool. “yes” means this feature is present, while “no” means this feature is not present.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Andes-enhanced performance monitoring feature is not supported.</td></tr>
<tr><td>1</td><td>Andes-enhanced performance monitoring feature is supported.</td></tr>
</table></td><td>RO</td><td>IM</td></tr>
</table>

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CCTLCSR</td><td>[16]</td><td>Indicates the presence of CSRs for CCTL operations.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Feature of CSRs for CCTL operations is not supported.</td></tr><tr><td>1</td><td>Feature of CSRs for CCTL operations is supported.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>EFHW</td><td>[17]</td><td>Indicates the support of <code>FLHW</code> and <code>FSHW</code> instructions.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td><code>FLHW</code> and <code>FSHW</code> instructions are not supported.</td></tr><tr><td>1</td><td><code>FLHW</code> and <code>FSHW</code> instructions are supported.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>VCCTL</td><td>[19:18]</td><td>Indicates the version number of CCTL command operation scheme supported by an implementation.</td><td>RO</td><td>IM</td></tr>
<tr><td>RVARCH</td><td>[52]</td><td>Indicates if <code>mrvarch_cfg</code> CSR is present or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td><code>mrvarch_cfg</code> CSR is not present.</td></tr><tr><td>1</td><td><code>mrvarch_cfg</code> CSR is present.</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

15.6.4 RISC-V Architecture Configuration Register

Mnemonic Name: mrvarch_cfg
IM Requirement: mmsc_cfg.RVARCH==1
Access Mode: Machine
CSR Address: 0xfca (non-standard read only)

64 … 4	3	2	1	0
0	Zbs	Zbc	Zbb	Zba

This register provides information regarding RISC-V Architecture. Note that the below table shows all valid values, but some of them may not be implemented.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Zba</td><td>[0]</td><td>Indicates the RISC-V Zba ISA extension is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Zba is not implemented.</td></tr><tr><td>1</td><td>Zba is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Zbb</td><td>[1]</td><td>Indicates the RISC-V Zbb ISA extension is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Zbb is not implemented.</td></tr><tr><td>1</td><td>Zbb is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Zbc</td><td>[2]</td><td>Indicates the RISC-V Zbc ISA extension is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Zbc is not implemented.</td></tr><tr><td>1</td><td>Zbc is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
<tr><td>Zbs</td><td>[3]</td><td>Indicates the RISC-V Zbs ISA extension is implemented or not.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Zbs is not implemented.</td></tr><tr><td>1</td><td>Zbs is implemented.</td></tr></table></td><td>RO</td><td>IM</td></tr>
</table>

15.7 Trigger Registers

15.7.1 Trigger Select

Mnemonic Name: tselect
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a0 (standard read/write)

63–0
TRIGGER_INDEX

This register determines which trigger is accessible through other trigger registers. The setting of accessible triggers must start at 0, and be contiguous. Writes of values greater than or equal to the number of supported triggers might result in a different value in this register than what was written. Debuggers should read back the value to confirm that what they wrote was a valid index.

Since triggers can be used by both Debug mode and Machine mode, the debugger must restore this register after the modification.

15.7.2 Trigger Data 1

Mnemonic Name: tdata1
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

63–60	59	58–0
TYPE	DMODE	DATA

This register provides access to the `tdata1` register of the currently selected trigger registers selected by the `tselect` register.

Field Name	Bits	Description	Type	Reset
DATA	[58:0]	Trigger-specific data	RW	0

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DMODE</td><td>[59]</td><td>Setting this field to indicate the trigger is used by Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td></tr>
<tr><td>1</td><td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>TYPE</td><td>[63:60]</td><td>Indicates the trigger type.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The selected trigger is invalid.</td></tr>
<tr><td>2</td><td>The selected trigger is an address/data match trigger.</td></tr>
<tr><td>3</td><td>The selected trigger is an instruction count trigger.</td></tr>
<tr><td>4</td><td>The selected trigger is an interrupt trigger.</td></tr>
<tr><td>5</td><td>The selected trigger is an exception trigger.</td></tr>
</table></td><td>RW</td><td>2</td></tr>
</table>

15.7.3 Trigger Data 2

Mnemonic Name: tdata2
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a2 (standard read/write)

This register provides accesses to the `tdata2` register of the currently selected trigger registers selected by the `tselect` register, and it holds trigger-specific data.

15.7.4 Trigger Data 3

Mnemonic Name: tdata3
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a3 (standard read/write)

This register provides access to the `tdata3` register of the currently selected trigger registers selected by the `tselect` register, and it holds trigger-specific data.

15.7.5 Trigger Info

Mnemonic Name: tinfo
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a4 (standard read/write)

63 … 16	15 … 0
0	INFO

This register provides accesses to the `tinfo` register of the currently selected trigger registers selected by the `tselect` register, and it indicates the supported trigger types of the currently selected trigger. Please see RISC-V External Debug Support Version 0.13 section 5.2 Trigger Registers for more information.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>INFO</td><td>[15:0]</td><td>One bit for each possible type in <code>tdata1</code>. Bit N corresponds to type N. If the bit is set, then that type is supported by the currently selected trigger. If the currently selected trigger does not exist, this field contains 1.
<table>
<tr><th>Bit N</th><th>Descriptions</th></tr>
<tr><td>0</td><td>When this bit is set, there is no trigger at this <code>tselect</code>.</td></tr>
<tr><td>1</td><td>Reserved and hardwired to 0.</td></tr>
<tr><td>2</td><td>When this bit is set, the selected trigger supports type of address/data match trigger.</td></tr>
<tr><td>3</td><td>When this bit is set, the selected trigger supports type of instruction count trigger.</td></tr>
<tr><td>4</td><td>When this bit is set, the selected trigger supports type of interrupt trigger.</td></tr>
<tr><td>5</td><td>When this bit is set, the selected trigger supports type of exception trigger.</td></tr>
<tr><td>15</td><td>When this bit is set, the selected trigger exists (so enumeration shouldn't terminate), but is not currently available.</td></tr>
<tr><td>Others</td><td>Reserved for future use.</td></tr>
</table></td><td>RO</td><td>0x003C</td></tr>
</table>

The detailed correlation between trigger types and Number of Trigger (Section 2.9.3) are as follows:

- INFO[2] = 1, all trigger types are supported.
- INFO[3] = 1, trigger 0 or 1 (`tselect` = 0 or 1) is supported.
- INFO[4] = 1, trigger 0 (`tselect` = 0) or trigger 4 (`tselect` = 4) is supported when Number of Triggers is 8.
- INFO[5] = 1,
  - trigger 1 (`tselect` = 1) is supported when Number of Triggers is 2,

- trigger 3 (`tselect` = 3) is supported when Number of Triggers is 4, or
- trigger 3 or 7 (`tselect` = 3 or 7) is supported when Number of Triggers is 8.

15.7.6 Trigger Control

Mnemonic Name: tcontrol
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a5 (standard read/write)

63 ... 8	7	6 ... 4	3	2 ... 0
0	MPTE	0	MTE	0

This register provides accesses to the `tcontrol` register, and it indicates the current native M-Mode debugging settings.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MTE</td><td>[3]</td><td>M-mode trigger enable field. When a trap into M-mode is taken, <code>MTE</code> is set to 0. When the MRET instruction is executed, <code>MTE</code> is set to the value of <code>MPTE</code>.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Triggers do not match/fire while the hart is in M-mode.</td></tr>
<tr><td>1</td><td>Triggers do match/fire while the hart is in M-mode.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>MPTE</td><td>[7]</td><td>M-mode previous trigger enable field. When a trap into M-mode is taken, <code>MPTE</code> is set to the value of <code>MTE</code>.</td><td>RW</td><td>0</td></tr>
</table>

15.7.7 Machine Context

Mnemonic Name: mcontext
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a8 (standard read/write)

63 … 6	5 … 0
0	MCONTEXT

This register provides access to the `mcontext` register.

Field Name	Bits	Description	Type	Reset
MCONTEXT	[5:0]	Machine mode software can write a context number to this register, which can be used to set triggers that only fire in that specific context.	RW	0

15.7.8 Match Control

Mnemonic Name: mcontrol
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

This register is accessible as `tdata1` when `TYPE` is 0 or 2.

63 … 60	59	58 … 53	52 … 16	15 … 12	11	10 … 7	6	5 4	3	2	1	0
TYPE	DMODE	MASKMAX	0	ACTION	CHAIN	MATCH	M	0	U	EXECUTE	STORE	LOAD

Field Name	Bits	Description	Type	Reset
LOAD	[0]	Setting this field to enable this trigger to compare virtual address of a load.	RW*	0
STORE	[1]	Setting this field to enable this trigger to compare virtual address of a store.	RW*	0
EXECUTE	[2]	Setting this field to enable this trigger to compare virtual address of an instruction.	RW	0
U	[3]	Setting this field to enable this trigger in U-mode.	RW	0
M	[6]	Setting this field to enable this trigger in M-mode.	RW	0

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MATCH</td><td>[10:7]</td><td>Setting this field to select the matching scheme.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Matches when the value equals <code>tdata2</code>.</td></tr>
<tr><td>1</td><td>Matches when the top M bits of the value match the top M bits of <code>tdata2</code>. M is 63 minus the index of the least-significant bit containing 0 in <code>tdata2</code>.</td></tr>
<tr><td>2</td><td>Matches when the value is greater than (unsigned) or equal to <code>tdata2</code>.</td></tr>
<tr><td>3</td><td>Matches when the value is less than (unsigned) <code>tdata2</code>.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>CHAIN</td><td>[11]</td><td>Setting this field to enable trigger chain.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>When this trigger matches, the configured action is taken.</td></tr>
<tr><td>1</td><td>While this trigger does not match, it prevents the trigger with the next index from matching.</td></tr>
</table>
If Number of Triggers is 2, this field is hardwired to 0 on trigger 1 (<code>tselect</code> = 1).
If Number of Triggers is 4, this field is hardwired to 0 on trigger 3 (<code>tselect</code> = 3).
If Number of Triggers is 8, this field is hardwired to 0 on trigger 3 and trigger 7 (<code>tselect</code> = 3 or 7).</td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ACTION</td><td>[15:12]</td><td>Setting this field to select what happens when this trigger matches.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Raise a breakpoint exception.</td></tr><tr><td>1</td><td>Enter Debug Mode. (Only supported when DMODE is 1.)</td></tr><tr><td>2</td><td>Raise a trace-on event to trace encoder.</td></tr><tr><td>3</td><td>Raise a trace-off event to trace encoder.</td></tr><tr><td>4</td><td>Raise a trace-notify event to trace encoder.</td></tr></table>When TRACE feature is not supported, trigger actions for trace will be illegal options.</td><td>RW</td><td>0</td></tr>
<tr><td>MASKMAX</td><td>[58:53]</td><td>Indicates the largest naturally aligned range supported by the hardware is 2ˆ12 bytes.</td><td>RO</td><td>12</td></tr>
<tr><td>DMODE</td><td>[59]</td><td>Setting this field to indicate the trigger is used by Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td></tr><tr><td>1</td><td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>TYPE</td><td>[63:60]</td><td>Indicates the trigger type.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The selected trigger is invalid.</td></tr><tr><td>2</td><td>The selected trigger is an address/data match trigger.</td></tr></table></td><td>RW</td><td>2</td></tr>
</table>

Note

The `LOAD/STORE` fields take no effect and are cleared if the `EXECUTE` field is set at the same time.

15.7.9 Instruction Count

Mnemonic Name: icount
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

This register is accessible as `tdata1` when `TYPE` is 3.

This register exists just for single-stepping support so `COUNT` is hardwired to 1. After this trigger fires, the mode bits (`M`, `U`) will be cleared instead of causing the `COUNT` bits to be decremented.

63–60	59	58–11	10	9	8–7	6	5–0
TYPE	DMODE	0	COUNT	M	0	U	ACTION

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ACTION</td><td>[5:0]</td><td>Setting this field to select what happens when this trigger matches.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Raise a breakpoint exception.</td></tr>
<tr><td>1</td><td>Enter Debug Mode. (Only supported when DMODE is 1.)</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>U</td><td>[6]</td><td>Setting this field to enable this trigger in U-mode.</td><td>RW</td><td>0</td></tr>
<tr><td>M</td><td>[9]</td><td>Setting this field to enable this trigger in M-mode.</td><td>RW</td><td>0</td></tr>
<tr><td>COUNT</td><td>[10]</td><td>This field is hardwired to 1 for single-stepping support</td><td>RO</td><td>1</td></tr>
<tr><td>DMODE</td><td>[59]</td><td>Setting this field to indicate the trigger is used by Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td></tr>
<tr><td>1</td><td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>TYPE</td><td>[63:60]</td><td>The selected trigger is an instruction count trigger.</td><td>RW</td><td>2</td></tr>
</table>

15.7.10 Interrupt Trigger

Mnemonic Name: itrigger
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

This register is accessible as `tdata1` when `TYPE` is 4.

This trigger may fire on any of the interrupts configurable in `mie`. The interrupts to fire on are configured by setting the same bit in `tdata2` as would be set in `mie` to enable the interrupt.

63–60	59	58–10	9	8–7	6	5–0
TYPE	DMODE	0	M	0	U	ACTION

Field Name	Bits	Description	Type	Reset
ACTION	[5:0]	Setting this field to select what happens when this trigger matches. Value / Meaning: 0: Raise a breakpoint exception. 1: Enter Debug Mode. (Only supported when `DMODE` is 1.)	RW	0
U	[6]	Setting this field to enable this trigger in U-mode.	RW	0
M	[9]	Setting this field to enable this trigger in M-mode.	RW	0
DMODE	[59]	Setting this field to indicate the trigger is used by Debug Mode. Value / Meaning: 0: Both Debug-mode and M-mode can write the currently selected trigger registers. 1: Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.	RW	0
TYPE	[63:60]	The selected trigger is an interrupt trigger.	RW	2

15.7.11 Exception Trigger

Mnemonic Name: etrigger
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a1 (standard read/write)

This register is accessible as `tdata1` when `TYPE` is 5.

This trigger may fire on up to XLEN of the Exception Codes defined in `mcause` (with Interrupt=0). Those causes are configured by writing the corresponding bit in `tdata2`. (E.g. to trap on an illegal instruction, the debugger sets bit 2 in `tdata2`.)

63 60	59	58 11	10	9	8 7	6	5 0
TYPE	DMODE	0	NMI	M	0	U	ACTION

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ACTION</td><td>[5:0]</td><td>Setting this field to select what happens when this trigger matches.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Raise a breakpoint exception.</td></tr><tr><td>1</td><td>Enter Debug Mode. (Only supported when DMODE is 1.)</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>U</td><td>[6]</td><td>Setting this field to enable this trigger in U-mode.</td><td>RW</td><td>0</td></tr>
<tr><td>M</td><td>[9]</td><td>Setting this field to enable this trigger in M-mode.</td><td>RW</td><td>0</td></tr>
<tr><td>NMI</td><td>[10]</td><td>Setting this field to enable this trigger in non-maskable interrupts, regardless of the values of u and m.</td><td>RW</td><td>0</td></tr>
<tr><td>DMODE</td><td>[59]</td><td>Setting this field to indicate the trigger is used by Debug Mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Both Debug-mode and M-mode can write the currently selected trigger registers.</td></tr><tr><td>1</td><td>Only Debug Mode can write the currently selected trigger registers. Writes from M-mode is ignored.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
TYPE	[63:60]	The selected trigger is an exception trigger.	RW	2

15.7.12 Trigger Extra

Mnemonic Name: textra
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug and Machine
CSR Address: 0x7a3 (standard read/write)

This register is accessible as `tdata3` when `TYPE` is 2, 3, 4, or 5 of the currently selected trigger registers selected by the `tselect` register, and it indicates the context matching scheme of the currently selected trigger.

63–57	56–51	50	49–11	10–2	1–0
0	MVALUE	MSELECT	0	SVALUE	SSELECT

Field Name	Bits	Description	Type	Reset
SSELECT	[1:0]	**Value** / **Meaning** 0: Ignore `MVALUE`. 1: This trigger will only match if the lower bits of `scontext` equal `SVALUE`. 2: This trigger will only match if `satp.ASID` equals `SVALUE`.	RW	0
SVALUE	[10:2]	Data used together with `SSELECT`.	RW	0
MSELECT	[50]	**Value** / **Meaning** 0: Ignore `MVALUE`. 1: This trigger will only match if the lower bits of `mcontext` equal `MVALUE`.	RW	0
MVALUE	[56:51]	Data used together with `MSELECT`.	RW	0

15.8 Debug and Trigger Registers

15.8.1 Debug Control and Status Register

Mnemonic Name: dcsr
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b0 (debug-mode-only)

15	14	13	12	11	10	9	8 – 6	5	4	3	2	1 – 0
EBREAKM	0	0	EBREAKU	STEPIE	STOPCOUNT	STOPTIME	CAUSE	0	MPRVEN	NMIP	STEP	PRV

63 – 32	31 – 28	27 – 16
0	XDEBUGVER	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>PRV</td><td>[1:0]</td><td>The privilege level that the hart was operating in when Debug Mode was entered. The external debugger can modify this value to change the hart's privilege level when exiting Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>User/Application</td></tr>
<tr><td>1</td><td>Reserved</td></tr>
<tr><td>2</td><td>Reserved</td></tr>
<tr><td>3</td><td>Machine</td></tr>
</table></td><td>RW</td><td>3</td></tr>
<tr><td>STEP</td><td>[2]</td><td>This bit controls whether non-Debug Mode instruction execution is in the single step mode. When set, the hart returns to Debug Mode after a single instruction execution. If the instruction does not complete due to an exception, the hart will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Single Step Mode is off</td></tr>
<tr><td>1</td><td>Single Step Mode is on</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>NMIP</td><td>[3]</td><td>When this bit is set, there is a Non-Maskable-Interrupt (NMI) pending for the hart. Since an NMI can indicate a hardware error condition, reliable debugging may no longer be possible once this bit becomes set.</td><td>RO</td><td>0</td></tr>
<tr><td>MPRVEN</td><td>[4]</td><td>This bit controls whether <code>mstatus.MPRV</code> takes effect in Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>MPRV in mstatus is ignored in Debug Mode.</td></tr>
<tr><td>1</td><td>MPRV in mstatus takes effect in Debug Mode.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>CAUSE</td><td>[8:6]</td><td>Reason why Debug Mode was entered. When there are multiple reasons to enter Debug Mode, the priority to determine the <code>CAUSE</code> value will be: trigger module > <code>EBREAK</code> > halt-on-reset > halt request > single step. Halt requests are requests issued by the external debugger.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>EBREAK</td></tr>
<tr><td>2</td><td>Trigger module</td></tr>
<tr><td>3</td><td>Halt request</td></tr>
<tr><td>4</td><td>Single step</td></tr>
<tr><td>5</td><td>Halt-on-reset</td></tr>
<tr><td>6–7</td><td>Reserved</td></tr>
</table></td><td>RO</td><td>0</td></tr>
</table>

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>STOPTIME</td><td>[9]</td><td>This bit controls whether timers are stopped in Debug Mode. The processor only drives its <code>stoptime</code> output pin to 1 if it is in Debug Mode and this bit is set. Integration effort is required to make timers in the platform observe this pin to really stop them.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not stop timers in Debug Mode</td></tr>
<tr><td>1</td><td>Stop timers in Debug Mode</td></tr>
</table></td><td>RW</td><td>1</td></tr>
<tr><td>STOPCOUNT</td><td>[10]</td><td>This bit controls whether performance counters are stopped in Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not stop counters in Debug Mode</td></tr>
<tr><td>1</td><td>Stop counters in Debug Mode</td></tr>
</table></td><td>RW</td><td>1</td></tr>
<tr><td>STEPIE</td><td>[11]</td><td>This bit controls whether interrupts are enabled during single stepping.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Disable interrupts during single stepping</td></tr>
<tr><td>1</td><td>Allow interrupts in single stepping</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>EBREAKU</td><td>[12]</td><td>This bit controls the behavior of <code>EBREAK</code> instructions in User/Application Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Generate a regular breakpoint exception</td></tr>
<tr><td>1</td><td>Enter Debug Mode</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>EBREAKM</td><td>[15]</td><td>This bit controls the behavior of EBREAK instructions in Machine Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Generate a regular breakpoint exception</td></tr><tr><td>1</td><td>Enter Debug Mode</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>XDEBUGVER</td><td>[31:28]</td><td>Version of the external debugger. 0 indicates that no external debugger exists and 4 indicates that the external debugger conforms to the RISC-V External Debug Support (TD003) V0.13.</td><td>RO</td><td>4</td></tr>
</table>

15.8.2 Debug Program Counter

Mnemonic Name: dpc
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b1 (debug-mode-only)

63 ... 0
DPC

When entering Debug Mode, the `dpc` CSR is updated with the virtual address of the next instruction to be executed. The behavior is described in more detail in Table 86. When leaving Debug Mode, the hart's `PC` is updated to the value stored in this register. The external debugger may write this register to change where the hart resumes.

Field Name	Bits	Description	Type	Reset
DPC	[63:0]	Debug Program Counter. Bit 0 is hardwired to 0.	RW	0

Table 86: Virtual Address in DPC upon Debug Mode Entry

Cause	Virtual Address in DPC
EBREAK	Address of the `EBREAK` instruction
single step	Address of the instruction that would be executed next if no debugging was going on.
trigger module	Address of the instruction which caused the trigger module to fire.

Continued on next page...

Table 86: (continued)

Cause	Virtual Address in DPC
halt request	Address of the next instruction to be executed at the time that Debug Mode was entered

15.8.3 Debug Scratch Register 0

Mnemonic Name: dscratch0
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b2 (debug-mode-only)

63–0
DSCRATCH0

A scratch register that is reserved for use by Debug Module.

15.8.4 Debug Scratch Register 1

Mnemonic Name: dscratch1
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7b3 (debug-mode-only)

63–0
DSCRATCH1

A scratch register that is reserved for use by Debug Module.

15.8.5 Exception Redirection Register

Mnemonic Name: dexc2dbg
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7e0 (non-standard read/write)

63 20	19	18 16	15	14	13	12	11	10 9	8	7	6	5	4	3	2	1	0
0	PMOV	0	BWE	SLPECC	ACE	HSP	MEC	0	UEC	SAF	SAM	LAF	LAM	NMI	II	IAF	IAM

This register redirects selected exceptions to cause the hart to enter Debug Mode instead of performing the standard trap handling.

When an exception is redirected to enter Debug Mode, the `dpc` CSR will be updated with the virtual address of the instruction causing the exception. The `dcsr.CAUSE` field will be updated with a value of 1 (EBREAK). The actual cause of the exception is saved to the `ddcause` CSR. The required updates to `mepc`, `mcause`, `mtval`, `mstatus`, and `mxstatus` CSRs for exceptions will not be affected by the redirection and these CSRs continue to provide information associated with the corresponding exceptions.

Field Name	Bits	Description	Type	Reset
IAM	[0]	Indicates whether Instruction Access Misaligned exceptions are redirected to enter Debug Mode. Value 0: Do not redirect Value 1: Redirect	RW	0
IAF	[1]	Indicates whether Instruction Access Fault exceptions are redirected to enter Debug Mode. Value 0: Do not redirect Value 1: Redirect	RW	0
II	[2]	Indicates whether Illegal Instruction exceptions are redirected to enter Debug Mode. Value 0: Do not redirect Value 1: Redirect	RW	0
NMI	[3]	Indicates whether Non-Maskable Interrupt exceptions are redirected to enter Debug Mode. Value 0: Do not redirect Value 1: Redirect	RW	0
LAM	[4]	Indicates whether Load Access Misaligned exceptions are redirected to enter Debug Mode. Value 0: Do not redirect Value 1: Redirect	RW	0

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>LAF</td><td>[5]</td><td>Indicates whether Load Access Fault exceptions are redirected to enter Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SAM</td><td>[6]</td><td>Indicates whether Store Access Misaligned exceptions are redirected to enter Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>SAF</td><td>[7]</td><td>Indicates whether Store Access Fault exceptions are redirected to enter Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UEC</td><td>[8]</td><td>Indicates whether U-mode Environment Call exceptions are redirected to enter Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>MEC</td><td>[11]</td><td>Indicates whether M-mode Environment Call exceptions are redirected to enter Debug Mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>HSP</td><td>[12]</td><td>Indicates whether Stack Protection exceptions are redirected to enter Debug Mode. This bit is present only when mmsc_cfg.HSP is set.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not redirect</td></tr><tr><td>1</td><td>Redirect</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ACE</td><td>[13]</td><td>Indicates whether ACE-related exceptions are redirected to enter Debug Mode. This bit is present only when <code>mmsc_cfg.ACE</code> is set.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not redirect</td></tr>
<tr><td>1</td><td>Redirect</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>SLPECC</td><td>[14]</td><td>Indicates whether local memory slave port ECC Error local interrupts are redirected to enter Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not redirect</td></tr>
<tr><td>1</td><td>Redirect</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>BWE</td><td>[15]</td><td>Indicates whether Bus-write Transaction Error local interrupts are redirected to enter Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not redirect</td></tr>
<tr><td>1</td><td>Redirect</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>PMOV</td><td>[19]</td><td>Indicates whether performance counter overflow interrupts are redirected to enter Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Do not redirect</td></tr>
<tr><td>1</td><td>Redirect</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

15.8.6 Debug Detailed Cause

Mnemonic Name: ddcause
IM Requirement: DEBUG_SUPPORT
Access Mode: Debug
CSR Address: 0x7e1 (non-standard read/write)

63 – 16	15 – 8	7 – 0
0	SUBTYPE	MAINTYPE

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MAINTYPE</td><td>[7:0]</td><td>Cause for redirection to Debug Mode.

Value	Meaning
0	Software Breakpoint (EBREAK)
1	Instruction Access Misaligned (IAM)
2	Instruction Access Fault (IAF)
3	Illegal Instruction (II)
4	Non-Maskable Interrupt (NMI)
5	Load Access Misaligned (LAM)
6	Load Access Fault (LAF)
7	Store Access Misaligned (SAM)
8	Store Access Fault (SAF)
9	U-mode Environment Call (UEC)
10–11	Reserved
12	M-mode Environment Call (MEC)
13–15	Reserved
16	Imprecise ECC error
17	Bus write transaction error
18	Performance Counter overflow
19–31	Reserved
32	Stack overflow exception
33	Stack underflow exception
34	ACE disabled exception
35–39	Reserved
40–47	ACE exception
$\geq$48	Reserved

</td><td>RO</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SUBTYPE</td><td>[15:8]</td><td>Subtypes for main type.
The table below lists the subtypes for DCSR.CAUSE==1 and DDCAUSE.MAINTYPE==3.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Illegal instruction</td></tr><tr><td>1</td><td>Privileged instruction</td></tr><tr><td>2</td><td>Non-existent CSR</td></tr><tr><td>3</td><td>Privilege CSR access</td></tr><tr><td>4</td><td>Read-only CSR update</td></tr></table></td><td>RO</td><td>0</td></tr>
</table>

15.9 User Trap Related CSRs

15.9.1 User Status

Mnemonic Name: ustatus
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x000 (standard read/write)

63 ... 5	4	3 ... 1	0
0	UPIE	0	UIE

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>UIE</td><td>[0]</td><td>U-mode interrupt enable bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UPIE</td><td>[4]</td><td>UPIE holds the value of the UIE bit prior to a trap.</td><td>RW</td><td>0</td></tr>
</table>

15.9.2 User Interrupt Enable

Mnemonic Name: uie
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x004 (standard read/write)

63–9	8	7–5	4	3–1	0
0	UEIE	0	UTIE	0	USIE

Field Name	Bits	Description	Type	Reset
USIE	[0]	U-mode software interrupt enable bit. **Value** / **Meaning** 0 / Disabled 1 / Enabled	RW	0
UTIE	[4]	U-mode timer interrupt enable bit. **Value** / **Meaning** 0 / Disabled 1 / Enabled	RW	0
UEIE	[8]	U-mode external interrupt enable bit. **Value** / **Meaning** 0 / Disabled 1 / Enabled	RW	0

15.9.3 User Trap Vector Base Address

Mnemonic Name: utvec
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x005 (standard read/write)

63 … 2	1 0
BASE[63:2]	0

This register determines the base address of the trap vector for U-mode trap handling. The least significant 2 bits are hardwired to zeros. When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

When `mmisc_ctl.VEC_PLIC` is 0 (PLIC is not in the vector mode), this register indicates the entry points for the trap handler and it may point to any 4-byte aligned location in the memory space.

On the other hand, when `mmisc_ctl.VEC_PLIC` is 1 (PLIC is in the vector mode), this register will be the base address of a vector table with 4-byte entries storing addresses pointing to interrupt service routines. And this register should be aligned to $2^{\log_2 N+2}$-byte boundary for PLIC with N interrupt sources. For example, if N is 1023, the minimum alignment requirement is 4096 bytes (4 KiB).

Field Name	Bits	Description	Type	Reset
BASE[63:2]	[63:2]	Base address for interrupt and exception handlers. See description above for alignment requirements when PLIC is in the vector mode.	RW	0

15.9.4 User Scratch Register

Mnemonic Name: uscratch
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x040 (standard read/write)

63 … 0
USCRATCH

A scratch register for temporary data storage, which is typically used by the U-mode trap handler.

Field Name	Bits	Description	Type	Reset
USCRATCH	[63:0]	Scratch register storage.	RW	0

15.9.5 User Exception Program Counter

Mnemonic Name: uepc
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x041 (standard read/write)

63 … 1	0
EPC	0

This register is written with the virtual address of the instruction that raises a trap and the trap is taken to U-mode.

Field Name	Bits	Description	Type	Reset
EPC	[63:1]	Exception program counter.	RW	0

15.9.6 User Cause Register

Mnemonic Name: ucause
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x042 (standard read/write)

63	62 … 10	9 … 0
INTERRUPT	0	EXCEPTION_CODE

This register indicates the cause of traps when they are taken to U-mode.

Field Name	Bits	Description	Type	Reset
EXCEPTION_CODE	[9:0]	Exception Code.	RW	0
INTERRUPT	[63]	Interrupt.	RW	0

Table 87: ucause Value After Trap

Interrupt	Exception Code	Description
1	0	User software interrupt
1	4	User timer interrupt
1	8	User external interrupt
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/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9-15	Reserved
0	32	Stack overflow exception
0	33	Stack underflow exception
0	40-47	Andes Custom Extension exception (see *Andes Custom Extension Specification* for more details)

15.9.7 User Trap Value

Mnemonic Name: utval
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x043 (standard read/write)

63 … 0
UTVAL

This register is updated when a trap is taken to U-mode. The updated value is dependent on the cause of traps:

- For hardware breakpoint exceptions, address-misaligned exceptions, or access-fault exceptions , it is the effective faulting addresses.
- For illegal instruction exceptions, the updated value is the faulting instruction. If the length of the instruction is less than XLEN bits long, the upper bits of `utval` are cleared to zero.
- For other exceptions, `utval` is set to zero.

For instruction-fetch access faults, this register will be updated with the address pointing to the portion of the instruction that caused the fault, while the `sepc` register will be updated with the address pointing to the beginning of the instruction.

When the width of the configured address is less than 64, the upper bits are hardwired to zeros.

Field Name	Bits	Description	Type	Reset
UTVAL	[63:0]	Exception-specific information for software trap handling.	RW	0

15.9.8 User Interrupt Pending

Mnemonic Name: uip
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x044 (standard read/write)

63–9	8	7–5	4	3–1	0
0	UEIP	0	UTIP	0	USIP

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>USIP</td><td>[0]</td><td>U-mode software interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>UTIP</td><td>[4]</td><td>U-mode timer interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>UEIP</td><td>[8]</td><td>U-mode external interrupt pending bit.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Not pending</td></tr><tr><td>1</td><td>Pending</td></tr></table></td><td>RO</td><td>0</td></tr>
</table>

15.9.9 User Detailed Trap Cause

Mnemonic Name: udcause
IM Requirement: misa[13]==1
Access Mode: User
CSR Address: 0x809 (non-standard read/write)

63 – 3	2 – 0
0	UDCAUSE

Field Name	Bits	Description	Type	Reset
UDCAUSE	[2:0]	This register further disambiguates causes of traps recorded in the `ucause` register. See the list below for details.	RW	0

The value of `UDCAUSE` for precise exception:

- When `ucause` == 1 (Instruction access fault)

Value	Meaning
0	Reserved
1	ECC/Parity error
2	PMP instruction access violation
3	Bus error
4	Reserved

- When `ucause` == 2 (Illegal instruction)

Value	Meaning
0	Please parse the `utval` CSR
1	FP disabled exception
2	ACE disabled exception

- When `ucause` == 5 (Load access fault)

Value	Meaning
0	Reserved
1	ECC/Parity error

Continued on next page...

Value	Meaning
2	PMP load access violation
3	Bus error
4	Misaligned address
5	Reserved
6	Reserved
7	Reserved

- When `ucause` == 7 (Store access fault)

Value	Meaning
0	Reserved
1	ECC/Parity error
2	PMP store access violation
3	Bus error
4	Misaligned address
5	Reserved
6	Reserved
7	Reserved

15.10 Memory and Miscellaneous Registers

15.10.1 Instruction Local Memory Base Register

Mnemonic Name: milmb
IM Requirement: ILM_SIZE_KB > 0
Access Mode: Machine
CSR Address: 0x7c0 (non-standard read/write)

63–10	9–4	3	2–1	0
IBPA	0	RWECC	ECCEN	IEN

This register controls instruction local memory.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>IEN</td><td>[0]</td><td>ILM enable control:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>ILM is disabled</td></tr>
<tr><td>1</td><td>ILM is enabled</td></tr>
</table></td><td>RO</td><td>1</td></tr>
<tr><td>ECCEN</td><td>[2:1]</td><td>Parity/ECC enable control:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Disable parity/ECC</td></tr>
<tr><td>1</td><td>Reserved</td></tr>
<tr><td>2</td><td>Generate exceptions only on unrepairable parity/ECC errors</td></tr>
<tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>RWECC</td><td>[3]</td><td>Controls diagnostic accesses of ECC codes of the ILM RAMs. When set, load/store to ILM reads/writes ECC codes to the <code>mecc_code</code> register. This bit can be set for injecting ECC errors to test the ECC handler.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Disable diagnostic accesses of ECC codes</td></tr>
<tr><td>1</td><td>Enable diagnostic accesses of ECC codes</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>IBPA</td><td>[63:10]</td><td>The base physical address of ILM. It has to be an integer multiple of the ILM size.</td><td>RO</td><td>ILM_ BASE[63:10]</td></tr>
</table>

15.10.2 Data Local Memory Base Register

Mnemonic Name: mdlmb
IM Requirement: DLM_SIZE_KB > 0
Access Mode: Machine
CSR Address: 0x7c1 (non-standard read/write)

63 — 10	9 — 4	3	2 — 1	0
DBPA	0	RWECC	ECCEN	DEN

This register controls data local memory.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DEN</td><td>[0]</td><td>DLM enable control:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>DLM is disabled</td></tr><tr><td>1</td><td>DLM is enabled</td></tr></table></td><td>RO</td><td>1</td></tr>
<tr><td>ECCEN</td><td>[2:1]</td><td>Parity/ECC enable control:<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC</td></tr><tr><td>1</td><td>Reserved</td></tr><tr><td>2</td><td>Generate exceptions only on unrepairable parity/ECC errors</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>RWECC</td><td>[3]</td><td>Controls diagnostic accesses of ECC codes of the DLM RAMs. When set, load/store to DLM reads/writes ECC codes to the <code>mecc_code</code> register. This bit can be set for injecting ECC errors to test the ECC handler.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable diagnostic accesses of ECC codes</td></tr><tr><td>1</td><td>Enable diagnostic accesses of ECC codes</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
DBPA	[63:10]	The base physical address of DLM. It has to be an integer multiple of the DLM size.	RO	DLM_ BASE[63:10]

15.10.3 ECC Code Register

Mnemonic Name: mecc_code
IM Requirement: mmsc_cfg.ECC == 1
Access Mode: Machine
CSR Address: 0x7c2 (non-standard read/write)

63–23	22	21–18	17	16	15–8	7–0
0	INSN	RAMID	P	C	0	CODE

This register is used for accessing ECC array.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CODE</td><td>[7:0]</td><td>This field records the ECC value on ECC error exceptions. This field is also used to read/write the ECC codes when diagnostic access of ECC codes are enabled (<code>milmb.RWECC</code>, <code>mdlmb.RWECC</code>, <code>mcache_ctl.IC_RWECC</code>, or <code>mcache_ctl.DC_RWECC</code> is 1).</td><td>RW</td><td>0</td></tr>
<tr><td>C</td><td>[16]</td><td>Correctable error. This bit is updated on parity/ECC error exceptions.<table><tr><td>Value</td><td>Meaning</td></tr><tr><td>0</td><td>Uncorrectable error</td></tr><tr><td>1</td><td>Correctable error</td></tr></table></td><td>RO</td><td>0</td></tr>
<tr><td>P</td><td>[17]</td><td>Precise error. This bit is updated on parity/ECC error exceptions.<table><tr><td>Value</td><td>Meaning</td></tr><tr><td>0</td><td>Imprecise error</td></tr><tr><td>1</td><td>Precise error</td></tr></table></td><td>RO</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>RAMID</td><td>[21:18]</td><td>The ID of RAM that caused parity/ECC errors. This bit is updated on parity/ECC error exceptions.
<table>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0–1</td><td>Reserved</td></tr>
<tr><td>2</td><td>Tag RAM of I-Cache</td></tr>
<tr><td>3</td><td>Data RAM of I-Cache</td></tr>
<tr><td>4</td><td>Tag RAM of D-Cache</td></tr>
<tr><td>5</td><td>Data RAM of D-Cache</td></tr>
<tr><td>6</td><td>Tag RAM of TLB</td></tr>
<tr><td>7</td><td>Data RAM of TLB</td></tr>
<tr><td>8</td><td>ILM</td></tr>
<tr><td>9</td><td>DLM</td></tr>
<tr><td>10–15</td><td>Reserved</td></tr>
</table></td><td>RO</td><td>0</td></tr>
<tr><td>INSN</td><td>[22]</td><td>Indicates if the parity/ECC error is caused by instruction fetch or data access.
<table>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>Data access</td></tr>
<tr><td>1</td><td>Instruction fetch</td></tr>
</table></td><td>RO</td><td>0</td></tr>
</table>

15.10.4 NMI Vector Base Address Register

Mnemonic Name: mnvec
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7c3 (non-standard read/write)

63 ... 0
MNVEC

This register indicates the entry point when an NMI occurs.

Field Name	Bits	Description	Type	Reset
MNVEC	[63:0]	Base address of the NMI handler. Its value is the zero-extended value of the `reset_vector[VALEN-1:0]` input signal to NX25(F).	RO	Pin Configured

15.10.5 Performance Throttling Control Register

Mnemonic Name: mpft_ctl
IM Requirement: POWERBRAKE_SUPPORT = "yes"
Access Mode: Machine
CSR Address: 0x7c5 (non-standard read/write)

63 … 9	8	7 … 4	3 … 0
0	FAST_INT	T_LEVEL	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>T_LEVEL</td><td>[7:4]</td><td>Throttling Level. The processor has the highest performance at throttling level 0 and the lowest performance at throttling level 15.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Level 0 (the highest performance)</td></tr>
<tr><td>1-14</td><td>Level 1-14</td></tr>
<tr><td>15</td><td>Level 15 (the lowest performance)</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>FAST_INT</td><td>[8]</td><td>Fast interrupt response. If this field is set, <code>mxstatus.PFT_EN</code> will be automatically cleared when the processor enters an interrupt handler.</td><td>RW</td><td>0</td></tr>
</table>

15.10.6 Cache Control Register

Mnemonic Name: mcache_ctl
IM Requirement: Cache optional (micm_cfg.ISZ != 0 or mdcm_cfg.DSZ != 0)
Access Mode: Machine
CSR Address: 0x7ca (non-standard read/write)

10	9	8	7	6	5 4	3 2	1	0
0	0	CCTL_SUEN	DC_RWECC	IC_RWECC	DC_ECCEN	IC_ECCEN	DC_EN	IC_EN

63 23	22 21	20 17	16 13	12	11
0	0	0	0	0	IC_FIRST_WORD

Field Name	Bits	Description	Type	Reset
IC_EN	[0]	Controls if the instruction cache is enabled or not. **Value** / **Meaning** 0 / I-Cache is disabled 1 / I-Cache is enabled	RW	0
DC_EN	[1]	Controls if the data cache is enabled or not. **Value** / **Meaning** 0 / D-Cache is disabled 1 / D-Cache is enabled	RW	0

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>IC_ECCEN</td><td>[3:2]</td><td>Parity/ECC error checking enable control for the instruction cache.
I-Cache is a clean cache and it can repair both correctable and uncorrectable parity/ECC errors by invalidating the affected lines and re-filling them from the next level memory. The only unrepairable errors by I-Cache is parity/ECC errors on locked cache lines. The processor assumes no bus accesses should be triggered for a locked line so parity/ECC errors cannot be repaired by re-filling from the next level memory and it consider such errors as fatal errors. Set this field to 2 to silently repair all repairable parity/ECC errors, and set it to 3 to take exceptions after repairing (invalidating) repairable parity/ECC errors. Exceptions will be taken for all unrepairable errors when this field is either 2 or 3.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC</td></tr><tr><td>1</td><td>Reserved</td></tr><tr><td>2</td><td>Generate exceptions only on unrepairable parity/ECC errors</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DC_ECCEN</td><td>[5:4]</td><td>Parity/ECC error checking enable control for the data cache.
In addition to repairing correctable parity/ECC errors, D-Cache can repair uncorrectable parity/ECC errors in clean cache lines by invalidating the affected lines and re-filling them from the next level memory. Only uncorrectable parity/ECC errors in dirty cache lines are unrepairable. Set this field to 2 silently repair all repairable parity/ECC errors, and set it to 3 to take exceptions after repairing (invalidating) repairable parity/ECC errors. Exceptions will be taken for all unrepairable errors when this field is either 2 or 3.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable parity/ECC</td></tr><tr><td>1</td><td>Reserved</td></tr><tr><td>2</td><td>Generate exceptions only on unrepairable parity/ECC errors</td></tr><tr><td>3</td><td>Generate exceptions on any type of parity/ECC errors</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>IC_RWECC</td><td>[6]</td><td>Controls diagnostic accesses of ECC codes of the instruction cache RAMs. It is set to enable CCTL operations to access the ECC codes (see Section 9.7). This bit can be set for injecting ECC errors to test the ECC handler.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable diagnostic accesses of ECC codes</td></tr><tr><td>1</td><td>Enable diagnostic accesses of ECC codes</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
DC_RWECC	[7]	Controls diagnostic accesses of ECC codes of the data cache RAMs. It is set to enable CCTL operations to access the ECC codes (see Section 9.7). This bit can be set for injecting ECC errors to test the ECC handler. **Value** / **Meaning**: 0 — Disable diagnostic accesses of ECC codes 1 — Enable diagnostic accesses of ECC codes	RW	0
CCTL_SUEN	[8]	Enable bit for User-mode software to access `ucctlbeginaddr` and `ucctlcommand` CSRs. **Value** / **Meaning**: 0 — Disable `ucctlbeginaddr` and `ucctlcommand` accesses in U mode 1 — Enable `ucctlbeginaddr` and `ucctlcommand` accesses in U mode	RW	0
IC_FIRST_WORD	[11]	I-Cache miss allocation filling policy **Value** / **Meaning**: 0 — Cache line data is returned critical (double) word first 1 — Cache line data is returned the lowest address (double) word first	RO	Configuration Dependent

15.10.7 Machine Miscellaneous Control Register

Mnemonic Name: mmisc_ctl
IM Requirement: Required
Access Mode: Machine
CSR Address: 0x7d0 (non-standard read/write)

63–10	9	8–7	6	5–4	3	2	1	0
0	0	0	MSA/UNA	ACES	BRPE	RVCOMPM	VEC_PLIC	0

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>VEC_PLIC</td><td>[1]</td><td>Selects the operation mode of PLIC:
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Regular mode</td></tr><tr><td>1</td><td>Vector mode</td></tr></table>
Please note that both this bit and the vector mode enable bit (<code>VECTORED</code>) of the Feature Enable Register in NCEPLIC100 should be turned on for the vectored interrupt support to work correctly. See Section 18.5.2 for the definition of the <code>VECTORED</code> bit.
This bit is hardwired to 0 if the vectored PLIC feature is not supported.</td><td>RW</td><td>0</td></tr>
<tr><td>RVCOMPM</td><td>[2]</td><td>RISC-V compatibility mode enable bit. If the compatibility mode is turned on, all Andes-specific instructions become reserved instructions.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>BRPE</td><td>[3]</td><td>Branch prediction enable bit. This bit controls all branch prediction structures.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled</td></tr><tr><td>1</td><td>Enabled</td></tr></table>This bit is hardwired to 0 if branch prediction structure is not supported.</td><td>RW</td><td>1</td></tr>
<tr><td>ACES</td><td>[5:4]</td><td>Andes Custom Extension (ACE) extension context status field:
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Off</td></tr><tr><td>1</td><td>Initial</td></tr><tr><td>2</td><td>Clean</td></tr><tr><td>3</td><td>Dirty</td></tr></table>

This field should not be Off (0) for ACE instructions to execute normally. ACE unit enters Off state by software program through <code>CSRW</code> instructions.
A normal flow to turn on ACE unit will be as follows:

<code>ACES</code> is in the Off state.
An ACE instruction executed in the Off state triggers an illegal instruction with <code>sdcause/mdcause</code> == 2 (ACE disabled exception).
The exception handler initializes all ACE register states, changes this field to the Initial state, and then returns from exception.
The ACE instruction is executed again. Since <code>ACES</code> is not in the Off state this time, it shall execute correctly. If any ACE register states are modified, <code>ACES</code> will be updated to the Dirty state automatically by hardware.

This field is hardwired to 0 if ACE extension is not configured.</td><td>RW</td><td>0</td></tr>
</table>

Continued on next page. . .

Field Name	Bits	Description	Type	Reset
MSA/UNA	[6]	This field controls whether the load/store instructions can access misaligned memory locations without generating Address Misaligned exceptions. Supported instructions: `LW/LH/LHU/SW/SH/LD/LWU/SD` <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Misaligned accesses generate Address Misaligned exceptions.</td></tr><tr><td>1</td><td>Misaligned accesses are allowed.</td></tr></table>	RW	IM

15.10.8 Machine CCTL Begin Address

Mnemonic Name: mcctlbeginaddr
IM Requirement: Cache optional
Access Mode: Machine
CSR Address: 0x7cb (non-standard read/write)

This register holds the address information required by CCTL operations. It is only present when (`micm_cfg.ISZ`!=0 or `mdcm_cfg.DSZ`!=0) and (`mmsc_cfg.CCTLCSR`==1).

63–0
VA

63–INDEX_MSB+3	INDEX_MSB+2–INDEX_MSB+1	INDEX_MSB–INDEX_LSB	INDEX_LSB-1–3	2–0
0	WAY	INDEX	OFFSET	0

- For "VA" type of CCTL operations:

  The `mcctlbeginaddr` register contains the starting virtual address for CCTL operations triggered by writes to the `mcctlcommand` register. For CCTL lock operations, the `mcctldata` register will be updated with a status value (0:fail, 1:success) when the operations complete.

  After an update to the `mcctlcommand` register with a VA-type command, the value of this register will be incremented with the byte size of the corresponding cache line.

- For "Index" type of CCTL operations:

  The `mcctlbeginaddr` register contains the cache index for CCTL operations triggered by writes to the `mcctlcommand` register.

  - For all Index-type commands other than "IX_RDATA" and "IX_WDATA":

    The `INDEX` field in this register will be incremented. If the incremented `INDEX` wraps around to 0 (i.e., the first way of a set), then the `WAY` field in this register will be incremented.

  - For "IX_RDATA" and "IX_WDATA" commands:

    The `OFFSET` field in this register will be incremented first to the next OFFSET value. If the incremented `OFFSET` field wraps around to 0 (i.e., the first double word of a cache line), then the `INDEX` field in this register will be incremented. If the incremented `INDEX` field wraps around to 0 (i.e., the first way of a set), then the `WAY` field in this register will be incremented.

15.10.9 Machine CCTL Command

Mnemonic Name: mcctlcommand
IM Requirement: Cache optional
Access Mode: Machine
CSR Address: 0x7cc (non-standard read/write)

Writing to this register will trigger a CCTL operation, with the type of the operation specified by the value written. Valid CCTL operations are defined in Table 88. See Section 9.7 for more information. CCTL operations are inherently not atomic, see notes below for usage limitations.

This register is only present when (`micm_cfg.ISZ`!=0 or `mdcm_cfg.DSZ`!=0) and (`mmsc_cfg.CCTLCSR`==1).

Table 88: CCTL Command Definition

	Value	Command	Type
0	0b00_000	L1D_VA_INVAL	VA
1	0b00_001	L1D_VA_WB	VA
2	0b00_010	L1D_VA_WBINVAL	VA
3	0b00_011	L1D_VA_LOCK	VA
4	0b00_100	L1D_VA_UNLOCK	VA
6	0b00_110	L1D_WBINVAL_ALL	-
7	0b00_111	L1D_WB_ALL	-
8	0b01_000	L1I_VA_INVAL	VA
11	0b01_011	L1I_VA_LOCK	VA
12	0b01_100	L1I_VA_UNLOCK	VA
16	0b10_000	L1D_IX_INVAL	Index
17	0b10_001	L1D_IX_WB	Index
18	0b10_010	L1D_IX_WBINVAL	Index
19	0b10_011	L1D_IX_RTAG	Index
20	0b10_100	L1D_IX_RDATA	Index
21	0b10_101	L1D_IX_WTAG	Index
22	0b10_110	L1D_IX_WDATA	Index
23	0b10_111	L1D_INVAL_ALL	-
24	0b11_000	L1I_IX_INVAL	Index
27	0b11_011	L1I_IX_RTAG	Index
28	0b11_100	L1I_IX_RDATA	Index

Continued on next page...

Table 88: (continued)

	Value	Command	Type
29	0b11_101	L1I_IX_WTAG	Index
30	0b11_110	L1I_IX_WDATA	Index

Note

CCTL operations take parameters from multiple CSR registers, thus they are inherently not atomic. In addition, the CSRs share common storage across privilege levels, making them vulnerable to be overwritten across context switches. This implies that higher privilege level software should backup the content of CCTL CSR registers with interrupts disabled before using them, and restore their values afterwards if CCTL operations will be invoked in multiple privilege levels. The same is true if CCTL operations will be used both in non-interrupt codes and in the interrupt handlers.

15.10.10 Machine CCTL Data

Mnemonic Name: mcctldata
IM Requirement: Cache optional
Access Mode: Machine
CSR Address: 0x7cd (non-standard read/write)

This register holds data required/returned by some CCTL operations. It is only present when (`micm_cfg.ISZ`!=0 or `mdcm_cfg.DSZ`!=0) and (`mmsc_cfg.CCTLCSR`==1). The complete list of CCTL operations are summarized in Table 89 and described below.

Table 89: CCTL Commands Which Access `mcctldata`

Value of **`mcctlcommand`**		**Command**	**Type**
3	0b00_011	L1D_VA_LOCK	VA
11	0b01_011	L1I_VA_LOCK	VA
19	0b10_011	L1D_IX_RTAG	Index
20	0b10_100	L1D_IX_RDATA	Index
21	0b10_101	L1D_IX_WTAG	Index
22	0b10_110	L1D_IX_WDATA	Index
27	0b11_011	L1I_IX_RTAG	Index
28	0b11_100	L1I_IX_RDATA	Index
29	0b11_101	L1I_IX_WTAG	Index
30	0b11_110	L1I_IX_WDATA	Index

- For CCTL lock operations: The `mcctldata` register will be updated with a status value (0: fail, 1:success) when the operations complete.
- For CCTL index read/write-data operations: `mcctldata[63:0]` holds the cache data for the operations.
- For CCTL index read/write-tag operations:
  - `mcctldata` register holds the cache tag for the operations. The register is as follows:

XLEN-1	XLEN-2	XLEN-3	XLEN-4	PALEN-10	PALEN-11	0
valid	lock	dirty/lock_dup	0		TAG=PA[(PALEN-1):10]	

– The TAG field does not hold every bits of PA[(PALEN-1):10] for the cache line. The cache tag RAMs do not hold all physical addresses down to bit 10. They only hold enough bits for tag look-ups. The unused lower order TAG bits will be reported as *Don't Care* value for tag-read operations and ignored on tag-write operations. The full PA value should be constructed using the corresponding index bits.

  * I-Cache TAG RAM holds PA[(PA_LEN-1):A], so `mcctldata[A-11:0]` are unused bits.

    · A = min(12:log_2(cache_size/way))

  * D-Cache TAG RAM holds PA[(PA_LEN-1):log_2(cache_size/way)], so `mcctldata[(`log_2`(cache_size/way)-11):0]` are unused bits.

– Bit XLEN-3 holds the dirty bit for D-Cache and duplicated lock (lock_dup) bit for I-Cache. The duplicate lock bit is for I-Cache to better tolerate soft-errors.

15.10.11 User CCTL Begin Address

Mnemonic Name: ucctlbeginaddr
IM Requirement: Cache optional
Access Mode: User and above
CSR Address: 0x80b (non-standard read/write)

This register is only present when \((micm_cfg.ISZ!=0 or mdcm_cfg.DSZ!=0) and mmsc_cfg.CCTLCSR==1 and misa[20]==1). It is an alias to the mcctlbeginaddr register and it is only accessible to User-mode software when mcache_ctl.CCTL_SUEN is 1. Otherwise illegal instruction exceptions will be triggered.

Note

- Both S-mode and U-mode software triggers CCTL operations through writing the ucctlcommand register; the associated addresses for CCTL operations are specified in the ucctlbeginaddr register.
- Due to the sharing of storage with mcctlbeginaddr, interrupt-handler/privileged-mode software should backup mcctlbeginaddr before use; see notes in Section 15.10.9 for usage limitations.

15.10.12 User CCTL Command

Mnemonic Name: ucctlcommand
IM Requirement: Cache optional
Access Mode: User and above
CSR Address: 0x80c (non-standard read/write)

Writing to this register will trigger a CCTL operation, with the type of the operation specified by the value written. Valid User CCTL commands are defined in Table 90.

This register is only present when \((`micm_cfg.ISZ`!=0 or `mdcm_cfg.DSZ`!=0) and `mmsc_cfg.CCTLCSR`==1 and `misa[20]`==1) and it is an alias to the `mcctlcommand` register. This register is only accessible to User-mode software when `mcache_ctl.CCTL_SUEN` is 1. Otherwise illegal instruction exceptions will be triggered.

Table 90: User CCTL Command Definition

Value of `ucctlcommand`		Command	Type
0	0b00_000	L1D_VA_INVAL	VA
1	0b00_001	L1D_VA_WB	VA
2	0b00_010	L1D_VA_WBINVAL	VA
8	0b01_000	L1I_VA_INVAL	VA

15.11 Hardware Stack Protection and Recording Registers

15.11.1 Machine Hardware Stack Protection Control

Mnemonic Name: mhsp_ctl
IM Requirement: STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)
Access Mode: Machine
CSR Address: 0x7C6 (non-standard read/write)

63–6	5	4	3	2	1	0
0	M	0	U	SCHM	UDF_EN	OVF_EN

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>OVF_EN</td><td>[0]</td><td>Enable bit for the stack overflow protection and recording mechanism. This bit will be cleared to 0 automatically by hardware when a stack protection (overflow or underflow) exception is taken.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The stack overflow protection and recording mechanism are disabled.</td></tr>
<tr><td>1</td><td>The stack overflow protection and recording mechanism are enabled.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>UDF_EN</td><td>[1]</td><td>Enable bit for the stack underflow protection mechanism. This bit will be cleared to 0 automatically by hardware when a stack protection (overflow or underflow) exception is taken.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The stack underflow protection is disabled.</td></tr>
<tr><td>1</td><td>The stack underflow protection is enabled.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SCHM</td><td>[2]</td><td>Selects the operating scheme of the stack protection and recording mechanism.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Stack overflow/underflow detection</td></tr><tr><td>1</td><td>Top-of-stack recording</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>U</td><td>[3]</td><td>Enables the <code>SP</code> protection and recording mechanism in User mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The mechanism is disabled in User mode.</td></tr><tr><td>1</td><td>The mechanism is enabled in User mode.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>M</td><td>[5]</td><td>Enables the <code>SP</code> protection and recording mechanism in Machine mode.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The mechanism is disabled in Machine mode.</td></tr><tr><td>1</td><td>The mechanism is enabled in Machine mode.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

15.11.2 Machine SP Bound Register

Mnemonic Name: msp_bound
IM Requirement: STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)
Access Mode: Machine
CSR Address: 0x7c7 (non-standard read/write)

63 … 0
MSP_BOUND

When the `SP` overflow detection mechanism is properly selected and enabled, any updated value to the `SP` register (via any instruction) is compared with the `msp_bound` register. If the updated value to the `SP` register is smaller than the `msp_bound` register, a stack overflow exception is generated. The stack overflow exception has an exception code of 32 in the `mcause` register.

When the top of stack recording mechanism is properly selected and enabled, any updated value to the `SP` register on any instruction is compared with the `msp_bound` register. If the updated value to the `SP` register is smaller than the `msp_bound` register, the `msp_bound` register is updated with this updated value. It is an RW-type register with the all-one reset value (0xFFFFFFFF for RV32 and 0xFFFFFFFFFFFFFFFF for RV64).

Programming Note:

- The “`CSRRW sp, msp_bound, rs`” instruction updates both `sp` and `msp_bound` registers at the same time. When the stack overflow detection mechanism is enabled, using this instruction may generate unpredictable exception behavior.

15.11.3 Machine SP Base Register

Mnemonic Name: msp_base
IM Requirement: STACKSAFE_SUPPORT = "yes" (mmsc_cfg.HSP == 1)
Access Mode: Machine
CSR Address: 0x7c8 (non-standard read/write)

63 … 0
SP_BASE

When the SP underflow detection mechanism is properly selected and enabled, any updated value to the SP register (via any instruction) is compared with the `msp_base` register. If the updated value to the SP register is greater than the `msp_base` register, a stack underflow exception is generated. The stack underflow exception has an exception code of 33 in the `mcause` register.

It is an RW-type register with the all-one reset value (0xFFFFFFFF for RV32 and 0xFFFFFFFFFFFFFFFF for RV64).

Programming Note:

- The "`CSRRW sp, msp_base, rs`" instruction updates both `sp` and `msp_base` registers at the same time. When the stack underflow detection mechanism is enabled, using this instruction may generate unpredictable exception behavior.

15.12 CoDense Registers

15.12.1 Instruction Table Base Address Register

Mnemonic Name: uitb
IM Requirement: CODENSE_SUPPORT = "yes"
Access Mode: User and above
CSR Address: 0x800 (non-standard read/write)

This register defines the base address of the CoDense instruction table. Each entry in the table contains a 32-bit instruction, which can be looked up and executed by a CoDense instruction. The table is typically generated by the compiler for replacing 32-bit instructions with the shorter 16-bit Andes CoDense instructions, hence reducing the code size.

63–2	1	0
ADDR	0	HW

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>HW</td><td>[0]</td><td>This bit specifies if the CoDense instruction table is hardwired.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The instruction table is located in memory. uitb.ADDR should be initialized to point to the table before using the CoDense instructions.</td></tr>
<tr><td>1</td><td>The instruction table is hardwired. Initialization of uitb.ADDR is not needed before using the CoDense instructions.</td></tr>
</table></td><td>RO</td><td>0</td></tr>
<tr><td>ADDR</td><td>[63:2]</td><td>The base address of the CoDense instruction table. This field is reserved if uitb.HW == 1.</td><td>RW</td><td>0</td></tr>
</table>

15.13 Physical Memory Protection Unit Configuration & Address Registers

15.13.1 PMP Configuration Registers

Mnemonic Name: pmpcfg0 and pmpcfg2
IM Requirement: PMP_SUPPORT
Access Mode: Machine
CSR Address: 0x3a0 and 0x3a2 (standard read/write)

0x3A0

31 24	23 16	15 8	7 0
PMP3CFG	PMP2CFG	PMP1CFG	PMP0CFG

0x3A1

63 56	55 48	47 40	39 32
PMP7CFG	PMP6CFG	PMP5CFG	PMP4CFG

0x3A2

31 24	23 16	15 8	7 0
PMP11CFG	PMP10CFG	PMP9CFG	PMP8CFG

0x3A3

63 56	55 48	47 40	39 32
PMP15CFG	PMP14CFG	PMP13CFG	PMP12CFG

PMP Configuration Format (for PMP$_i$CFG)

7	6 5	4 3	2	1	0
L	0	A	X	W	R

Field Name	Bits	Description	Type	Reset
R	[0]	Read access control. **Value** / **Meaning** 0: Read accesses are not allowed. 1: Read accesses are allowed.	RW	0
W	[1]	Write access control. **Value** / **Meaning** 0: Write accesses are not allowed. 1: Write accesses are allowed.	RW	0

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>X</td><td>[2]</td><td>Instruction execution control.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Instruction execution is not allowed.</td></tr><tr><td>1</td><td>Instruction execution is allowed.</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>A</td><td>[4:3]</td><td>Address matching mode.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>OFF: Null region.</td></tr><tr><td>1</td><td>TOR: Top of range. For PMP entry 0, it matches any address A < <code>pmpaddr0</code>. For PMP entry i, it matches any address A such that $\texttt{pmpaddr}_i$ > A >= $\texttt{pmpaddr}_{i-1}$. But the 4-byte range is not supported.</td></tr><tr><td>2</td><td>Reserved.</td></tr><tr><td>3</td><td>NAPOT: Naturally aligned power-of-2 region. This mode makes use of the low-order bits of the associated address register to encode the size of the range. See Table 91 for range encoding from the value of a PMP address register. The minimal size of NAPOT regions must be 8 bytes.</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>L</td><td>[7]</td><td>Write lock and permission enforcement bit for Machine mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Machine mode writes to PMP entry registers are allowed. R/W/X permissions apply to U modes.</td></tr>
<tr><td>1</td><td>For PMP entry i, writes to PMP$_i$CFG and PMPADDR$_i$ are ignored. Additionally, if PMP$_i$CFG.A is set to TOR, writes to pmpaddr$_{i-1}$ are ignored as well. As for permission enforcement, R/W/X permissions apply to all modes. This bit can only be cleared to 0 with a system reset.</td></tr>
</table></td><td>W1S*</td><td>0</td></tr>
</table>

Note

The register type of the L field is W1S because only a system reset can clear this bit.

15.13.2 PMP Address Register

Mnemonic Name: pmpaddr0–pmpaddr15
IM Requirement: PMP_SUPPORT
Access Mode: Machine
CSR Address: 0x3b0 to 0x3bf (standard read/write)

63–54	53–0
0	PMPADDR[55:2]

Each PMP address register encodes bits 55–2 of a 56-bit physical address, as shown in the register format. Different address matching mode also decides the way how PMP addressing and memory size.

If PMP configuration field A is 1, the address matching mode becomes TOR (Top of range) mode. At TOR mode, PMP entry 0 matches any address of A < pmpaddr0. PMP entry i matches any address of pmpaddr(i-1) <= A < pmpaddr(i), where i ranges from 1 and 15. When i is 1, PMP entry 1 matches any address of pmpaddr0 <= A < pmpaddr1. The start address of PMP entry 1 is bit[55:2] of pmpaddr0, the end address is (bit[55:2]-1) of pmpaddr1 and the Memory size is (end address - start address + 1)*4 bytes.

If PMP configuration field A is 3, address matching mode becomes NAPOT (Naturally aligned power-of-2 region) mode. At NAPOT mode, not all physical address bits may be implemented. The encoding is described in Table 91, where "a" is an arbitrary value representing one bit value of the physical address.

Table 91: NAPOT Range Encoding in PMP Address and Configuration Registers

Register Content	Match Size(Byte)
aaaa...aaa0	8
aaaa...aa01	16
aaaa...a011	32
...	...
aa01...1111	2^{XLEN}
a011...1111	2^{XLEN+1}
0111...1111	2^{XLEN+2}
1111...1111	2^{XLEN+3} [*1]

Note

1. The behavior of this register content is the same as that of 0111...1111 and hence the match size is equivalent to 2^{XLEN+2}.

The smallest PMP entry granularity is 8-bytes and pmpaddr$_i$[0] is hardwired to zero when the mode is OFF or TOR.

When PMP is used in the cacheable memory space, a PMP region must also naturally align to the size of the cache line. Otherwise, a deliberate load to a cache line partially covered by a PMP region may bring the full cache line to the Data Cache and allow CCTL operations to access the reset of the cache line that may have a different access restriction.

16 Instruction Throughput and Latency

This chapter lists instruction throughput and latency. The instruction throughput is the number of cycles before executing the next independent instruction of the same kind. The instruction latency is the number of cycles before executing the next instruction with read-after-write dependency.

16.1 ALU Instructions

The latency and the throughput of ALU instructions are both 1 cycle. ALU instructions include:

- Add/Sub: ADD, SUB, ADDI, ADDW, SUBW, ADDIW
- Shift: SLL, SRL, SRA, SLLI, SRLI, SRAI, SLLW, SRLW, SRAW, SLLIW, SRLIW, SRAIW
- Logical: AND, OR, XOR, ANDI, ORI, XORI
- Compare: SLT, SLTU, SLTI, SLTIU
- LUI and AUIPC
- Load effective address instructions
- ADDIGP
- String processing: FFB, FFZMISM, FFMISM, FLMISM
- Bit field operation: BFOS, BFOZ
- RISC-V bit-manipulation extension instructions

16.2 Load Instructions

The throughput and latency of load instructions are summarized in the following table.

Table 92: Load Instruction Throughput and Latency

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
load word/dword from DLM/D-Cache	1	2
load word/dword from ILM	2	4
load word/dword from AXI/AHB	4*	5*
load byte/halfword from DLM/D-Cache	1	3
load byte/halfword from ILM	2	4

Continued on next page...

Table 92: (continued)

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
load byte/halfword from AXI/AHB	4*	6*
load word/dword from low access-latency AHB	3*	3*
load byte/halfword from low access-latency AHB	3*	4*

Note

1. The calculation of latency should take system delay into consideration.

16.3 Multiply Instructions

The latency and throughput of multiply instructions depend on the multiplier implementation.

Table 93: Multiply Instruction Throughput and Latency: Radix Multiplier

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
MULHU	4 + 64 / LOG2(MUL_RADIX)	6 + 64 / LOG2(MUL_RADIX)
MUL, MULH, MULHSU	5 + 64 / LOG2(MUL_RADIX)	7 + 64 / LOG2(MUL_RADIX)

Table 94: Multiply Instruction Throughput and Latency: Fast Multiplier

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
MUL, MULH, MULHU, MULHSU	1	3

16.4 Divide and Remainder Instructions

The divide and remainder instructions are implemented using the non-restoring division algorithm with early termination detection.

Table 95: Divide Instruction Throughput and Latency

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
DIVU, REMU, DIVUW, REMUW	7–73	7–70

Continued on next page...

Table 95: (continued)

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
DIV, REM	7–73	8–71

16.5 Branch and Jump Instruction

The branch and jump instruction throughput is 1 cycle/instruction. Branch mis-prediction penalty is 3 cycles.

16.6 Trap Return Instruction

The trap return instruction flushes the entire pipeline, and the penalty is 5 cycles.

16.7 FENCE Instruction

`FENCE` instructions serve to flush the entire pipeline and waits for outstanding memory accesses to complete. In cases where there are no outstanding memory accesses, a penalty of 6 cycles is incurred. However, when consecutive `FENCE` instructions are issued, the latter one may have fewer penalties in cycle count.

16.8 Scalar Floating-Point Instructions

The instruction latencies of Floating-point instruction groups are shown in the following table.

Table 96: Scalar Floating-Point Instruction Throughput and Latency

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
FADD.x, FSUB.x, FMUL.x, FMADD.x, FMSUB.x, FNMADD.x FNMSUB.x	1	4
FDIV.S, FSQRT.S	19	19
FDIV.D, FSQRT.D	33	33
FLW, FSW, FLD, FSD	1	3
FSGNJ.x, FSGNJN.x, FSGNJX.x, FMIN.x, FMAX.x	1	2
FCLASS.x, FEQ.x, FLT.x, FLE.x	1	3

Continued on next page...

Table 96: (continued)

Instruction	Throughput (Cycles/Instruction)	Latency (Cycles)
FCVT.W.S, FCVT.WU.S, FCVT.L.S, FCVT.LU.S, FCVT.L.D, FCVT.LU.D	4	3
FCVT.S.W, FCVT.S.WU, FCVT.S.L, FCVT.S.LU, FCVT.D.L, FCVT.D.LU	1	4
FCVT.D.S, FCVT.S.D	1	4
FMV.X.W, FMV.X.D	1	3
FMV.W.X, FMV.D.X	1	1

16.9 ACE Instructions

Latencies of ACE instructions are custom-defined. They depend on the complexity of the specified operations.

Two additional cycles are needed when there is GPR dependency between an ACE instruction and its subsequent instruction.

17 AE350 Platform

The AE350 platform is a pre-integrated reference platform in Verilog implementing the AE350 memory map that contains an NX25(F) processor. AE350 supports two frameworks: AHB and AXI. The framework type can be configured through the **Bus Type** configuration option. The corresponding block diagrams are depicted respectively in Figure 2 and Figure 4. The peripheral platform IPs may be available as either unencrypted RTL or encrypted RTL depending on the NX25(F) licensing agreements.

17.1 I/O Signals

The top-level module of this platform is ae350_chip. I/O signals of ae350_chip are described in Table 97. Signal types are listed below:

Term	Description
I	Input signals
O	Output signals
I/O	Bi-directional signals

Table 97: I/O Signals

Interface	Signal Name	Type	Description
General	X_om	I	Operation mode
	X_aopd_por_b	I	Power-on reset in the always-on power domain
	X_por_b	I	Power-on reset in the main power domain
	X_hw_rstn	I	Hardware reset
	X_oschio	I/O	High frequency oscillator output
	X_oschin	I	High frequency oscillator input
	X_osclio	I/O	Low frequency oscillator output
	X_osclin	I	Low frequency oscillator input
	X_mpd_pwr_off	O	Main power domain power off indication
	X_rtc_wakeup	O	Alarm wake-up event
	X_wakeup_in	I	External wake-up event
SPI 1	X_spi1_clk	I/O	SPI clock
	X_spi1_csn	I/O	SPI chip select (Active-Low)
	X_spi1_mosi	I/O	SPI bus master output / slave input
	X_spi1_miso	I/O	SPI bus master input / slave output
	X_spi1_holdn	I/O	SPI hold (Active-Low)
	X_spi1_wpn	I/O	SPI WP (Active-Low)
SPI 2	X_spi2_clk	I/O	SPI Clock
	X_spi2_csn	I/O	SPI chip select (Active-Low)
	X_spi2_mosi	I/O	SPI bus master output / slave input
	X_spi2_miso	I/O	SPI bus master input / slave output
	X_spi2_holdn	I/O	SPI hold (Active-Low)
	X_spi2_wpn	I/O	SPI WP (Active-Low)
I2C	X_i2c_scl	I/O	I^2C clock

Continued on next page...

Table 97: (continued)

Interface	Signal Name	Type	Description
	X_i2c_sda	I/O	I^2C data
JTAG	X_tdi	I	Test data input*
	X_tdo	O	Test data output*
	X_tms	I	Test mode select*
	X_tck	I	Test clock*
	X_trst	I	Test reset*
GPIO	X_gpio[31:0]	I/O	General purpose I/O
UART 1	X_uart1_rxd	I	UART1 serial data input
	X_uart1_txd	O	UART1 serial data output
	X_uart1_ctsn	I	UART1 modem clear to send (Active-Low)
	X_uart1_rtsn	O	UART1 modem request to send (Active-Low)
	X_uart1_dcdn	I	UART1 modem data carrier detect (Active-Low)
	X_uart1_dsrn	I	UART1 modem data set ready (Active-Low)
	X_uart1_dtrn	O	UART1 modem data terminal ready (Active-Low)
	X_uart1_out1n	O	UART1 user-defined output 1 (Active-Low)
	X_uart1_out2n	O	UART1 user-defined output 2 (Active-Low)
	X_uart1_rin	I	UART1 modem ring indicator (Active-Low)
UART 2	X_uart2_rxd	I	UART2 serial data input
	X_uart2_txd	O	UART2 serial data output
	X_uart2_ctsn	I	UART2 modem clear to send (Active-Low)
	X_uart2_rtsn	O	UART2 modem request to send (Active-Low)
	X_uart2_dcdn	I	UART2 modem data carrier detect (Active-Low)
	X_uart2_dsrn	I	UART2 modem data set ready (Active-Low)
	X_uart2_dtrn	O	UART2 modem data terminal ready (Active-Low)
	X_uart2_out1n	O	UART2 user-defined output 1 (Active-Low)
	X_uart2_out2n	O	UART2 user-defined output 2 (Active-Low)
	X_uart2_rin	I	UART2 modem ring indicator (Active-Low)
PWM	X_pwm_ch0	O	PWM channel 0
	X_pwm_ch1	O	PWM channel 1
	X_pwm_ch2	O	PWM channel 2
	X_pwm_ch3	O	PWM channel 3

Note

- All JTAG ports will be removed when macro `PLATFORM_NO_DEBUG_SUPPORT` is defined.
- JTAG ports X_tdi, X_tdo, and X_trst will be removed when macro `AE350_JTAG_TWOWIRE` is defined even if `PLATFORM_NO_DEBUG_SUPPORT` is not defined.

17.2 Clock Generation

The clock tree in the reference platform is illustrated in Figure 16. Clocks are generated inside instance *ae350_aopd.ae350_clkgen* (module ae350_clkgen) in the always-on power domain. The clock ratios between various clocks are generated according to the SMU setup.

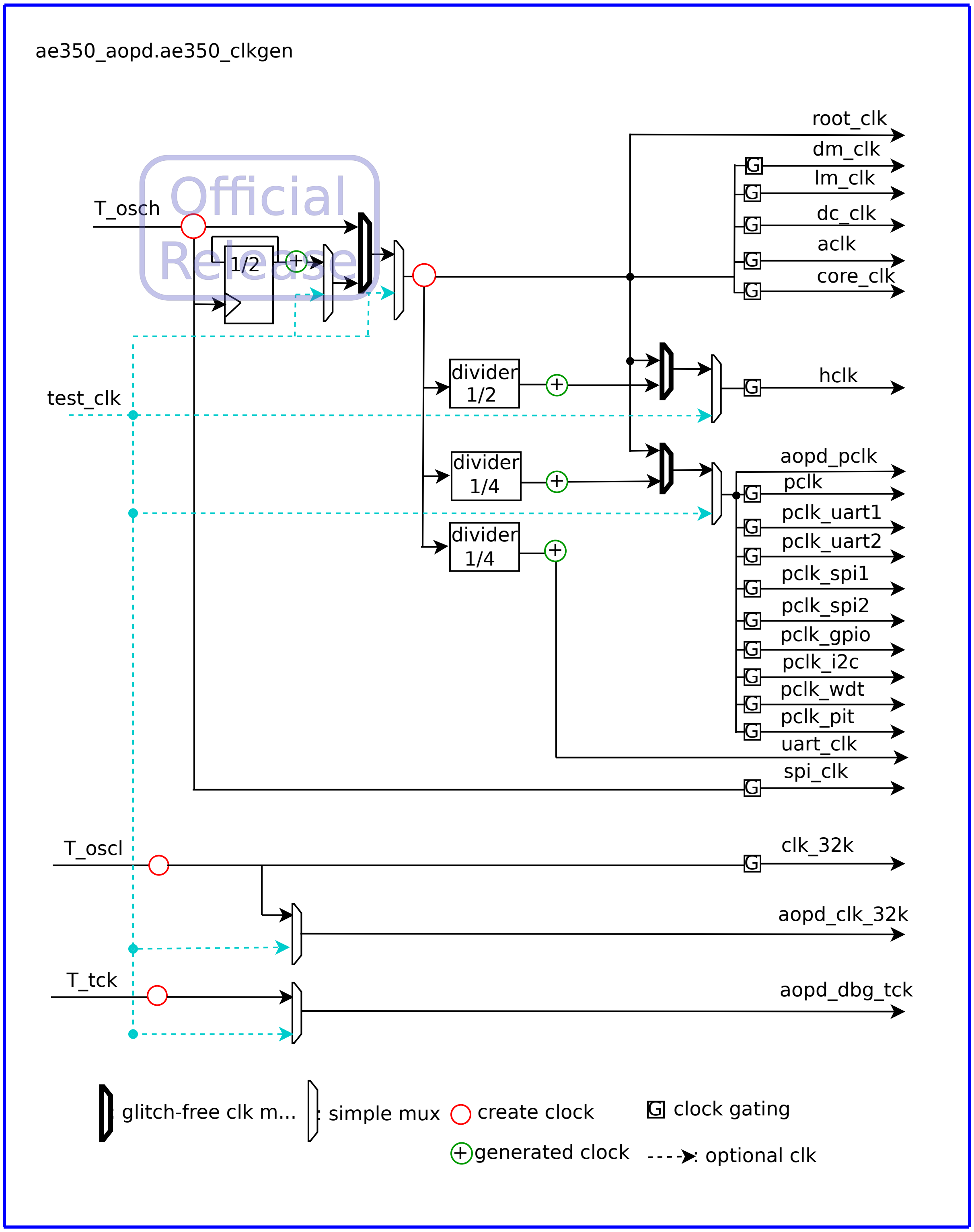


Figure 16: AE350 Clock Tree

In the clock generators, all sources of clock domains are multiplexed with the test clock, **test_clk**, for DFT. The clock generators need three clock sources: **OSCH**, **OSCL**, and **T_tck**.

Table 98: Clock Sources

Clock Source	Description
OSCH	High-frequency oscillator clock. (Also known as `T_osch` in the design)
OSCL	Low-frequency oscillator clock. OSCL is usually 32.768KHz for RTC. (Also known as `T_oscl` in the design)
T_tck	Clock source from the ICE box.

The clock generators produce the following clocks for the platform.

Table 99: Generated Clocks

Generated Signal	Signal Description
core_clk	Processor clock = OSCH or OSCH/2.
aclk	AXI bus clock = core_clk.
hclk	AHB bus clock = core_clk or core_clk/2.
lm_clk	A clock source for local memory clock.
dc_clk	A clock source for D-Cache clock.
pclk	APB bus clock = core_clk or core_clk/4.
pclk_uart1	APB bus clock for UART1 controller.
pclk_uart2	APB bus clock for UART2 controller.
pclk_spi1	APB bus clock for SPI1 controller.
pclk_spi2	APB bus clock for SPI2 controller.
pclk_gpio	APB bus clock for GPIO controller.
pclk_i2c	APB bus clock for I2C controller.
pclk_pit	APB bus clock for PIT controller.
uart_clk	A clock source for the UART controller. It must be independent of the bus clock ratio.
spi_clk	An independent clock source for the SPI SCLK divider logic in the SPI controller.
dm_clk	A clock source to synchronize relevant reset signals for PLDM.
clk_32k	Low-frequency clock for WDT controller, GPIO controller, pit controller, etc.
dbg_tck	A clock source for JTAG debug port clock.
aopd_pclk	APB bus clock for AOPD applications.
aopd_clk_32k	Low-frequency clock for RTC and AOPD applications.

Additionally, when SYNTHESIS is defined, the clock tree is as shown in Figure 17.

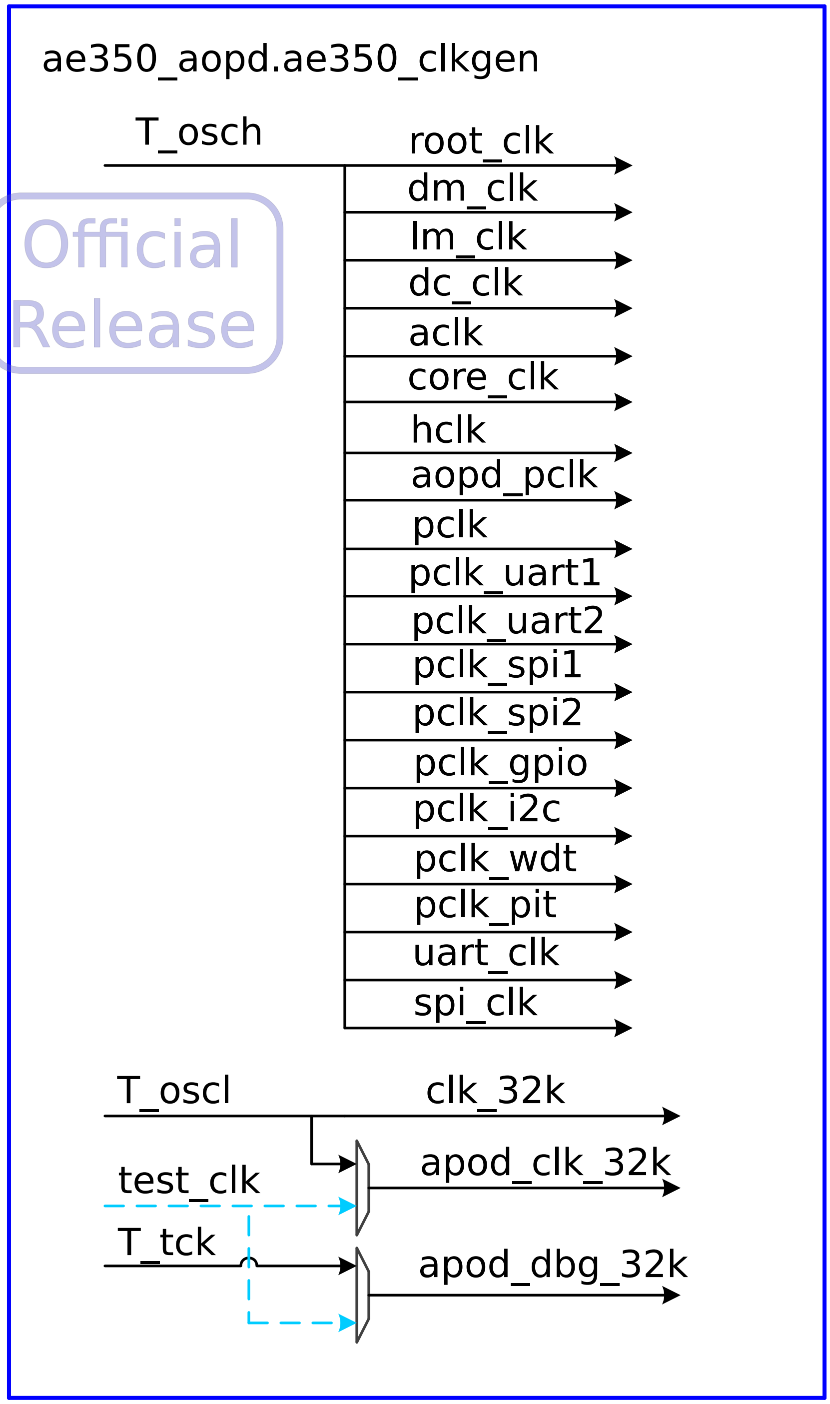


Figure 17: AE350 Clock Tree with SYNTHESIS define

However, the clock tree should be modified for FPGA implementations due to the limitations of clock resources in FPGA. The clock tree for the FPGA application is illustrated in Figure 18. Users can modify them for various FPGA platforms. BUFGCTRL (or BUFG mostly) in the figure is the most commonly used clock routing resource in Xilinx FPGA.

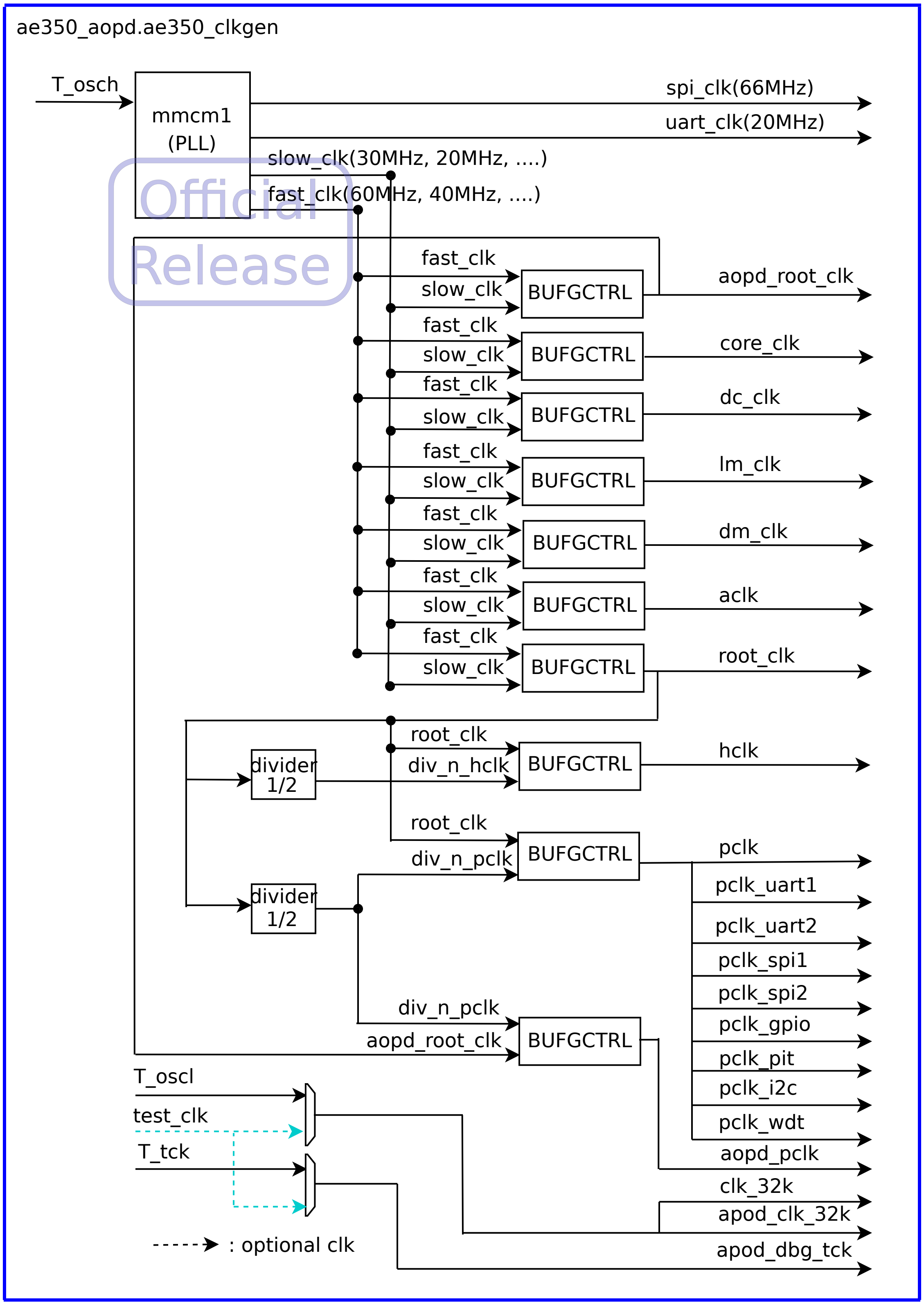


Figure 18: AE350 FPGA Clock Tree

17.3 Reset Generation

The reset tree in the reference platform is illustrated in Figure 19. Most resets are generated inside instance *ae350_rstgen* (module ae350_rstgen) in the always-on power domain. All resets in this section are Active-Low.

In the system, all sources of reset signals are multiplexed with a test reset, **test_rstn**, for DFT. The reset generators include the following reset sources highlighted in red in Figure 19.

Table 100: AE350 Reset Sources

Reset Source	Description
T_aopd_por_b	Power-on reset in the always-on power domain
T_por_b	Power-on reset in the main power domain
T_hw_rstn	Hardware reset
wdt_rstn	Watch-dog reset
pcs*x*_reset	Please refer to PCS_reset in Power Control Slot section.

The reset generator synchronizes all reset signals according to their respective clock domains. The generated reset signals are highlighted in blue in Figure 19.

Table 101: AE350 Generated Reset Signals

Generated Reset Signal	Description
rtc_rstn	RTC reset
aopd_por_prstn	AOPD power-on reset synchronized to aopd_pclk.
aopd_por_rstn	AOPD power-on reset synchronized to aopd_dbg_tck.
aopd_por_dbg_rstn	AOPD power-on reset and ndmreset synchronized to aopd_pclk.
por_b_psync	Power-on reset synchronized to pclk.
por_rstn	Power-on reset.
main_rstn	Reset signal used for the clock generator.
main_rstn_csync	Reset signal synchronized to the processor clock for the clock generator.
uart_rstn	UART reset for the uart_clk domain.
spi_rstn	SPI reset for the spi_clk domain.
presetn	APB bus reset
hresetn	AHB bus reset
aresetn	AXI bus reset
core*n*_resetn	Processor *n* reset.

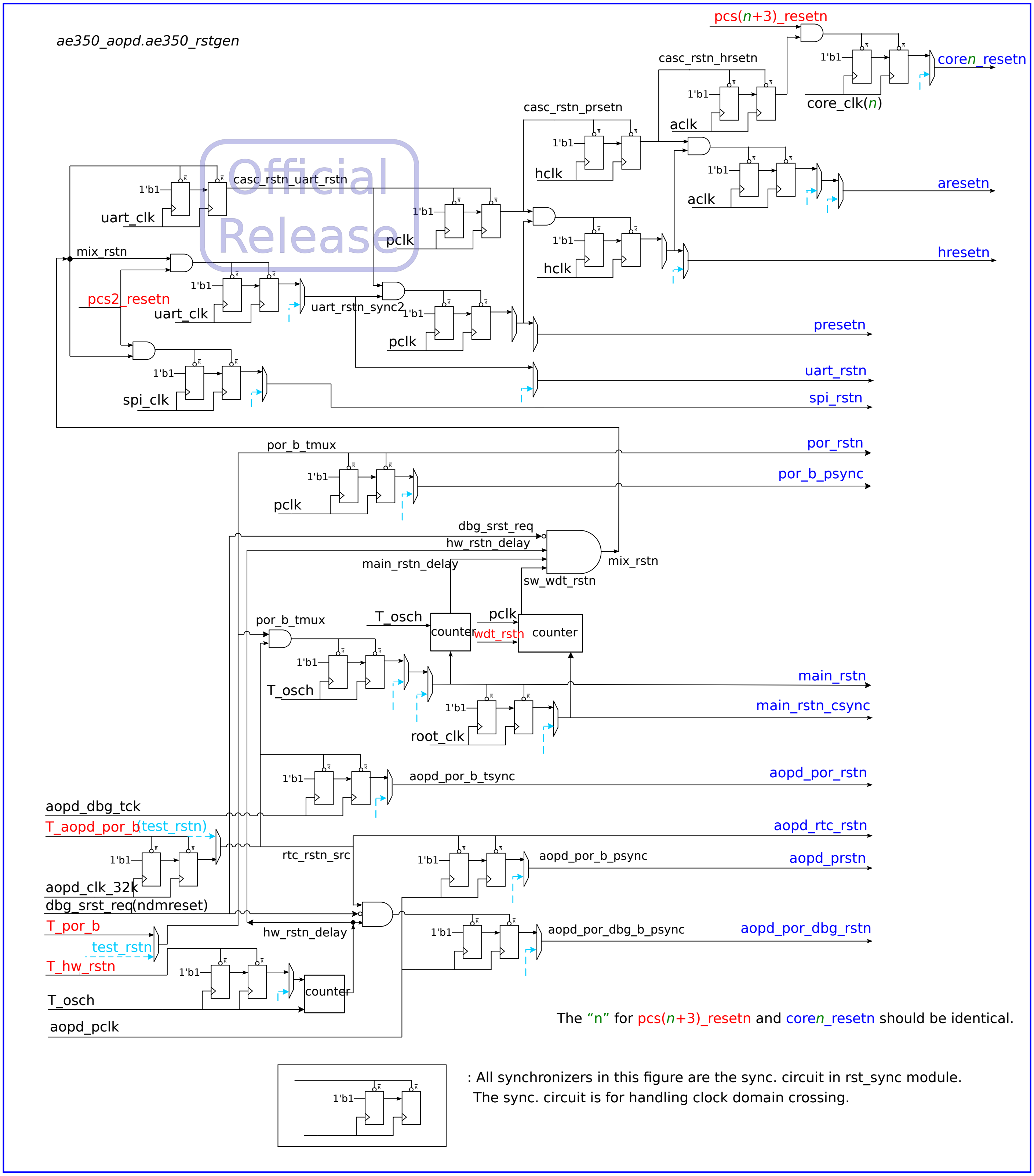


Figure 19: AE350 Reset Tree

17.4 AE350 Memory Map

The default memory map is shown in Table 102. This map is based on the default Device Region configuration. If the Device Region configuration is modified, this map should be adjusted accordingly so that the spaces of I/O devices still reside in the device region. If any peripheral IP is not configured in Section 2, the related device region will not exist.

Table 102: AE350 Memory Map

Address		Description
Begin	End	
0x00000000	0x003FFFFF	RAM Bridge
0x80000000	0x87FFFFFF	SPI1 AHB Memory
0xA0000000	0xA01FFFFF	Local Memory Slave Port: ILM
0xA0200000	0xA03FFFFF	Local Memory Slave Port: DLM
0xC0000000	0xC00FFFFF	BMC
0xE0000000	0xE00FFFFF	AHB Decoder
0xE4000000	0xE43FFFFF	PLIC
0xE6000000	0xE60FFFFF	Machine Timer
0xE6400000	0xE67FFFFF	PLIC-SWINT
0xE6800000	0xE68FFFFF	Debug Module
0xF0000000	0xF00FFFFF	APBBRG
0xF0100000	0xF01FFFFF	SMU
0xF0200000	0xF02FFFFF	UART1
0xF0300000	0xF03FFFFF	UART2
0xF0400000	0xF04FFFFF	PIT
0xF0500000	0xF05FFFFF	WDT
0xF0600000	0xF06FFFFF	RTC
0xF0700000	0xF07FFFFF	GPIO
0xF0A00000	0xF0AFFFFF	I2C
0xF0B00000	0xF0BFFFFF	SPI1
0xF0C00000	0xF0CFFFFF	DMAC
0xF0F00000	0xF0FFFFFF	SPI2
0xF2000000	0xF20FFFFF	DTROM

Continued on next page...

Table 102: (continued)

Address		Description
Begin	End	
0x100000000	0x1FFFFFFFF	Uncacheable alias to region 0x00000000 - 0xFFFFFFFF. The data in cacheable regions will be cached into L1 caches of the processor. This region is an uncacheable alias to the cacheable regions. Accesses to this region will bypass the L1 caches. This is useful when the processor and external bus masters need to share data in the same memory space. Andes RISC-V Linux port requires this region to be present. This region is present only when BIU_ADDR_WIDTH is 33 bits.

Note

- The RAM bridge space indicates the size of the RAM behind this bridge. It can be different from the size of the address space allocated to the bridge on the bus. The default setting allocates a 2GiB space (0x00000000 – 0x7FFFFFFF) to the bridge on the bus. When the bridge sees a transaction for addresses outside of the addressable RAM, it will return an error response.
- In addition to the bus view described here, ILM/DLM are accessible by the processor through private address spaces visible only to the processor:
  - The address ranges of these private address spaces are controlled by `milmb.IBPA` and `mdlmb.DBPA`.
  - The private address spaces have higher priority than the bus address spaces in the processor. Accesses will be directed to go through the local memory interfaces and bypass the bus address spaces if they hit the private address spaces.
  - If overlapping of address spaces is not desired, the `milmb` and `mdlmb` control registers could be programmed to avoid overlapping.
  - ILM is visible to the processor at 0x00000000 in the default setting.
  - DLM is visible to the processor at 0x00200000 in the default setting.
  - Debug Module region will be mapped to the default slave when macro `PLATFORM_NO_DEBUG_SUPPORT` is defined

17.5 Interrupt Assignment

Interrupts in a RISC-V platform are classified into two types: local interrupts and global interrupts. Local interrupts are interrupts that go directly into a RISC-V processor, and global interrupts get arbitrated through a platform-level interrupt controller (PLIC) before going into the RISC-V processor as the *external interrupts*.

Local interrupts supported by each core include non-maskable interrupts (`nmi`), machine timer interrupts (`mtip`) and software interrupts (`msip`). Additionally, external interrupt pins (`meip` and `seip`) accept arbitrated interrupt signaling from PLIC.

Table 103 summarizes the interrupt source connectivity.

In the AE350 platform, the PLIC module is instantiated a second time with all interrupt sources tied to zero as the software interrupt controller (`PLIC_SW`). The capability of the PLIC controller to generate interrupts through its programming registers is used for generating software interrupts.

The global interrupt sources in the AE350 platform and their connectivity to PLIC are summarized in Table 104.

If any peripheral IP is not configured in Section 2, the related interrupt signal will be tied to 0.

Table 103: NX25(F) Interrupt Assignment

Interrupt Signal	Description
nmi	WDT
mtip	Machine Timer
meip	PLIC
seip	PLIC
msip	PLIC_SW

Table 104: PLIC Interrupt Source

Interrupt Source	Description
1	RTC period
2	RTC alarm
3	PIT
4	SPI1
5	SPI2
6	IIC
7	GPIO

Continued on next page...

Table 104: (continued)

Interrupt Source	Description
8	UART1
9	UART2
10	DMAC
11	BMC*
26	SMU standby_req
27	SMU wakeup_ok

Note

- Only used in the AHB framework.

17.6 DMA Hardware Handshake ID

The source/destination IDs are needed for programming the handshake pairs when initiating DMA transfers. Table 105 assigns the handshake ID for all devices of the AE350 platform. The source and destination ID should not be the same for a handshake pair since the DMA controller does not support transferring data back to the same device.

Table 105: DMA Hardware Handshake ID

DMA Handshake ID	Devices
0	SPI1 TX
1	SPI1 RX
2	SPI2 TX
3	SPI2 RX
4	UART1 TX
5	UART1 RX
6	UART2 TX
7	UART2 RX
8	I2C

17.7 Platform IP Functional Description

17.7.1 ATCAPBBRG100 – AHB-to-APB Bridge

The AHB-to-APB bridge translates AHB transactions to APB transactions targeting a specific APB slave according to the slave base address and address space size configurations. Features of the bridge include:

- Compliant with AMBA 4 APB
- Support of 24/32 bits address width
- Support of up to 32 APB slaves
  - Including one internal slave for slave information registers and register programming
- Configurable base/size for each downstream APB slave
- Support of various synchronous AHB/APB clock ratios (N:1, N = 1, 2, 3, . . .)
- Support of write buffering

17.7.2 ATCAXI2AHB100 – AXI-to-AHB Synchronous Bridge

ATCAXI2AHB100 is a protocol converter that converts the AMBA AXI4 protocol to the AMBA AHB-Lite protocol. It enables the AXI4 master to access AHB-Lite slave devices. Features of the AXI-to-AHB synchronous bridge include:

- Compliant with AMBA AXI4
- Compliant with AMBA AHB-Lite
- Support of 24–64 bits address width
- Support of 32/64 bits data width
- Single clock and reset domains for both AXI4 and AHB-Lite interfaces
- Same address and data widths between AXI4 and AHB-Lite interfaces
- Support of narrow transfers

17.7.3 ATCAXI2AHB200 – AXI-to-AHB Asynchronous Bridge

ATCAXI2AHB200 is a protocol converter that translates the AMBA AXI4 protocol to the AMBA AHB-Lite protocol. Features of the AXI-to-AHB asynchronous bridge include:

- Compliant with AMBA AXI4
- Compliant with AMBA AHB-Lite
- Support of 24–64 bits address width
- Support of 32/64 data width
- Asynchronous clock and reset domains between AXI4 and AHB-Lite interfaces

- Support of narrow transfers

17.7.4 ATCBMC200 – AHB Bus Matrix

The AHB bus matrix controller (BMC) provides a backbone for the AE350 AHB framework, which allows multiple simultaneous bus transactions. In addition, the controller has slave information registers for software to enumerate the AHB bus components. Features of ATCBMC200 include:

- Support of AMBA AHB-Lite
- Support of 24/32 bits address width
- Support of 32/64/128/256 bits data width
- Support of up to 16 AHB-Lite masters
- Support of up to 16 AHB slaves
- Configurable connectivity between masters and slaves
- Programmable two-level priority arbitration scheme

17.7.5 ATCBMC300 – AXI Bus Matrix

The AXI bus matrix (BMC) provides a backbone for the AE350 AXI framework. All functional units are connected through the AXI bus matrix. Features of ATCBMC300 include:

- Compliant with AMBA AXI4
- Support of 24–64 bits address width
- Support of 32/64/128/256 bits data width
- Support of configurable AxID width on master ports
  - A single configurable width for all masters
  - The width of AxID on slave ports is this width plus 4
- Support of up to 16 AXI masters
- Support of up to 32 AXI slaves
  - Including one internal slave for slave information registers and register programming
- Configurable connectivity between masters and slaves
- Configurable number of outstanding transactions
- 3 programmable priority levels with round-robin arbitration

17.7.6 ATCBUSDEC200 – AHB Bus Decoder

ATCBUSDEC200 is an AMBA AHB-Lite decoder. It receives bus transactions from the upstream port and dispatches the transactions to the downstream ports according to the slave base address and space size configurations. This decoder also provides slave information registers for software to look up the memory space assignment information. Features of the AHB bus decoder include:

- Compliant with AMBA AHB-Lite
- One upstream AHB-Lite port
- Support of 24/32–64 bits address width
- Support of 32/64/128/256 bits data width
- Support of up to 32 downstream AHB-Lite slaves
  - Slave 0 is reserved as the internal slave for slave information registers
- Configurable base/size for each downstream AHB-Lite slave

17.7.7 ATCBUSDEC350 – AXI Bus Decoder

ATCBUSDEC350 is the duplicated version of ATCBMC300, except all words (despite the case) in the source code that contain "ATCBMC300" are renamed to "ATCBUSDEC350". ATCBUSDEC350 uses its own specific configuration (`atcbusdec350_config.vh`), which configures just one master.

17.7.8 ATCDMAC110 – DMA Controller

The Direct Memory Access Controller (DMAC) enhances system performance by transferring large data blocks between devices in the background to offload the processor. Features of DMAC include:

- Compliant with AMBA 2 AHB and APB4
- Up to 8 configurable DMA channels
- Up to 32 DMA request/acknowledge for hardware handshake
- Group round-robin arbitration scheme with 2 priority levels
- Support of 8, 16, and 32-bit data transfers
- Support of 24 – 64 bits address width
- Support of chain transfers

17.7.9 ATCDMAC300 – DMA Controller

The Direct Memory Access Controller (DMAC) enhances system performance by transferring large data blocks between devices in the background to offload the processor. Features of DMAC include:

- Compliant with AMBA AXI4 and APB4

- Support of up to 8 DMA channels
- Support of up to 16 DMA request/acknowledge pairs for hardware handshake
- Support of up to two AXI master ports for data transfers
- Support of up to two configurable DMA cores
- Support of an APB slave port for DMA register programming
- Support of 24–64 bits AXI address width
- Support of 32/64/128/256 bits AXI data width
- Support of narrow transfers on the AXI bus
- Support of group round-robin arbitration scheme with 2 priority levels
- Support of chain transfers

17.7.10 ATCGPIO100 – GPIO Controller

The General Purpose I/O (GPIO) controller supports up to 32 channels with independently programmable input/output control. Features of the GPIO controller include:

- Support of up to 32 GPIO channels (pins)
- Independent control of each channel
- Programmable I/O direction
- Optional pull-up/down control
- Optional support of interrupt trigger control
- Flexible combination of interrupt trigger modes: high/low level trigger and rising/falling/both edge trigger
- Optional de-bounce functionality for input channels

17.7.11 ATCIIC100 – I2C Controller

The I2C controller handles communications to the Inter-Integrated Circuit (IIC or I2C) serial interface. Features of the I2C controller include:

- Programmable to be either a master or a slave device
- Programmable clock/data timing
- Support of the I2C-bus Standard-mode (100 kb/s), Fast-mode (400 kb/s) and Fast-mode plus (1 Mb/s)
- Support of hardware handshaking to the DMA controller
- Support of the master-transmit, master-receive, slave-transmit and slave-receive modes
- Support of the multi-master mode
- Support of 7-bit and 10-bit addressing modes
- Support of general call addressing mode
- Support of auto clock stretch

17.7.12 ATCPIT100 – PIT Controller

The Programmable Interval Timer (PIT) controller is a set of compact multi-function timers, which can be used as pulse width modulators (PWM) or simple timers. Each multi-function timer provides the following 6 usage scenarios:

- One 32-bit timer
- Two 16-bit timers
- Four 8-bit timers
- One 16-bit PWM
- One 16-bit timer and one 8-bit PWM
- Two 8-bit timers and one 8-bit PWM

Features of the PIT controller include:

- Support of AMBA 2.0 APB bus protocol
- Support of up to 4 multi-function timers
- Six usage scenarios (combination of timer and PWM) for each multi-function timer
- Programmable source of timer clock

17.7.13 ATCRAMBRG200 – RAM Bridge

The RAM bridge controller allows standard SRAMs to be accessed on the AHB bus. Features of the RAM bridge include:

- Support of 10–64 bits address width
- Support of 32/64/128/256/512/1024 bits data width

17.7.14 ATCRAMBRG300 – RAM Bridge

The RAM bridge controller allows standard SRAMs to be accessed on the AXI bus. Features of the RAM bridge include:

- Support of 24–64 bits address width
- Support of 32/64/128/256 bits data width
- Support of exclusive accesses

17.7.15 ATCRTC100 – Real-Time Clock

Real-time clock (RTC) keeps track of current time relative to a base time. The time is stored in a RTC counter which records the amount of elapsed time since RTC is enabled. Features of RTC include:

- The frequency of clock source (before the clock divider) for the counter is 32.768KHz.
- Separate second, minute, hour and day counters.
- Periodic interrupts: half-second, second, minute, hour and day interrupts.
- Programmable alarm interrupt with specified second, minute and hour values.

RTC duplicates in functionality the RISC-V Machine Timer (Section 19). The RTC module is offered for compatibility of existing Andes platform software environment. It may be configured out for a pure RISC-V platform.

17.7.16 Sample_dtrom – Device Tree ROM

sample_dtrom is a read-only device on the APB bus to hold the binary form of the Device Tree description. Device Tree description is a standard way for Linux and embedded software to discover platform level configurations. This device is not enabled by default and `AE350_DTROM_SUPPORT` should be defined to enable this device.

It is okay to build a platform without this device as software platforms can usually take alternative device tree information from other media. However, built-in device tree information in the hardware lessens the burden to manage hardware variations.

The ROM data is expected to be provided by file `sample_dtrom.data` located in the working directory of synthesis tools.

A script **$NDS_HOME**/andes_ip/scripts/gen_dtrom.pl is provided to automatically generate a device tree description and binary based on ae350 platform configuration settings. It will save the generated description in the `ae350.dts` file and the compiled binary in `sample_dtrom.data`. The Device Tree Compiler `dtc` should be in the search path to run this script.

17.7.17 ATCSIZEDN100 – AHB Downsizer

The ATCSIZEDN100 bridge converts transactions between the upstream AMBA AHB-Lite bus of wider data width and the downstream AMBA AHB-Lite bus of narrower data width. Features of the AHB downsizer include:

- Compliant with AMBA AHB-Lite
- Support of 24–64 bits address width
- Support of 128-to-64, 64-to-32, 128-to-32 data width conversion
- Configurable write data buffering

17.7.18 ATCSIZEDN300 – AXI Downsizer

The ATCSIZEDN300 bridge converts transactions between the upstream AMBA AXI4 bus of wider data width and the downstream AMBA AXI4 bus of narrower data width. Features of the AXI downsizer include:

- Compliant with AMBA AXI4
- Support of 24–64 bits address width
- Support of 4–32 bits ID width
- Support of 256-to-32, 256-to-64, 256-to-128, 128-to-32, 128-to-64, 64-to-32 data width conversion

17.7.19 ATCSIZEUP300 – AXI Upsizer

The ATCSIZEUP300 bridge converts transactions between the upstream AMBA AXI4 bus of narrower data width to the downstream AMBA AXI4 bus of wider data width. Features of the AXI upsizer include:

- Compliant with AMBA AXI4
- Support of 24–64 bits address width
- Support of 4–32 bits AxID width
- Configurable data buffering
- Configurable data width conversion:
  - 32 to 64, 128 and 256 bits
  - 64 to 128 and 256 bits
  - 128 to 256 bits

17.7.20 ATCSPI200 – SPI Controller

The SPI controller handles communications to the Serial Peripheral Interface (SPI). The supported serial data formats range from 4 bits to 32 bits in length. Features of the SPI controller include:

- Compliant with AMBA 2 AHB protocol specification
- Compliant with AMBA 3 APB protocol specification
- Support of MSB/LSB first transfer
- Support of Direct Memory Access (DMA) data transfer
- Support of programmable SPI SCLK
- Support of memory-mapped access (read-only) through AHB bus or EILM bus
- Support of SPI slave mode
- Configurable Dual and Quad I/O SPI interfaces
- Configurable TX/RX FIFO depth (The depth could be 2, 4, 8, 16, 32, 64, or 128)

- Configurable programming port location on AHB/APB/EILM interfaces

17.7.21 ATCUART100 – UART Controller

The UART controller handles communications to the Universal Asynchronous Receiver/Transmitter (UART) serial interface. It has the following features:

- Compatible with the 16C550A register structure
- Support of the hardware flow control (CTS/RTS)
- Support of hardware handshaking to the DMA controller
- Option of by-8 or by-16 over-sampling frequency
- Support of 16/32/64/128-entry transmit/receive FIFO depth

17.7.22 ATCWDT200 – Watchdog Timer

The Watchdog Timer (WDT) controller prevents the system from hanging if software is trapped in a deadlock condition. A decrementing counter (the watchdog timer) is maintained in WDT and a watchdog interrupt will be generated once the watchdog timer reaches zero. The timer should be restarted in the watchdog interrupt service routine. A secondary timer called system reset timer starts ticking after the watchdog interrupt, and it gets canceled upon restart of the watchdog timer. Should the watchdog timer be not restarted in time after the watchdog interrupt is triggered, system reset will be triggered by the system reset timer to reset the system. Features of WDT include:

- Internal/external clock source selection
- Separate timers for the watchdog interrupt and the system reset
  - Eight choices of watchdog timer intervals
  - Four choices of reset timer intervals
- Register write protection for watchdog timer control register and restart register
  - Configurable magic number for register write protection
  - Configurable magic number to restart the watchdog timer

17.8 Duplicated Copies of Platform IPs

In AE350, some IP modules need to be instantiated more than once but with different macro settings. To avoid the macro name conflict, one module is duplicated to another module. For example, atcmstmux300 is duplicated from atcbmc300 with all occurrences of string "atcbmc300" inside all related file names and file contents changed to "atcmstmux300". Current duplicated modules are listed below.

New Module Name	Duplicated From
atcbusdec200_rom	ATCBUSDEC200
atcmstmux300	ATCBMC300
atcbmc300_1	ATCBMC300

Furthermore, the ae350_cpu_subsystem design additionally instantiates modules vc_bmch, vc_busdech, vc_bmcx, vc_busdecx, and/or vc_slvport_busdech. Those modules are not only duplicated but also specialized versions of other IPs as listed in the following table.

New Module Name	Specialized From
vc_bmch	ATCBMC200
vc_busdech	ATCBUSDEC200
vc_bmcx	ATCBMC300
vc_busdecx	ATCBUSDEC300 [*1]
vc_slvport_busdech	ATCBUSDEC200

Note

1. ATCBUSDEC300 is an AMBA AXI decoder but is not included in the NX25(F) release package. Contact Andes for more information.

17.9 IP Configurations

Find the configuration files of peripheral IPs under the following directory:

$NDS_HOME/andes_ip/ae350/define/

The configuration file name is **${IP_NAME}**_config.vh. In the release package, two configuration files can be found for each peripheral IP. The effective one is located under the above mentioned directory and it is specialized for AE350. The other one is located under **$NDS_HOME**/andes_ip/peripheral_ip/**$IP_NAME**/hdl/include/, which is a generic version provided by each individual IP for reference only and is not used by the AE350 platform. For example, for ATCGPIO100, two configuration files with the same name exist as:

$NDS_HOME/andes_ip/ae350/define/atcgpio100_config.vh
$NDS_HOME/andes_ip/peripheral_ip/atcgpio100/hdl/include/atcgpio100_config.vh

Only the configuration files under **$NDS_HOME**/andes_ip/ae350/define/ take effect since this directory is listed first with the +incdir+ option in the file list input to the simulator and synthesizer. However, the paths **$NDS_HOME**/andes_ip/peripheral_ip/**$IP_NAME**/hdl/include/ are still present in the file list as the +incdir+ options. The purpose is to specify the location of the **${IP_NAME}**_const.vh files, which contain constant/non-configurable macro settings for each IP. Please see the respective data sheets of the component IPs for configuration details.

17.10 Platform Configurations

The platform-level configurations are defined in the platform configuration file:

$NDS_HOME/andes_ip/ae350/top/hdl/include/ae350_config.vh

Configurations defined in ae350_config.vh are listed as follows:

Table 106: AE350 Configuration Options

Macro Name	Description
PLATFORM_JTAG_TWOWIRE	See Section 2.10.1.
PLATFORM_PLDM_SYS_BUS_ACCESS	See Section 2.10.2.
PLATFORM_PLDM_PROGBUF_SIZE	See Section 2.10.3.
PLATFORM_PLDM_HALTGROUP_COUNT	See Section 2.10.4.

The platform debug related options should not be modified manually since these options are automatically generated and overwritten by the configuration tool.

17.11 System Management Unit

ATCSMU100 provides versatile system management capabilities, including clock, reset and power control based on power domain partitions. The ATCSMU100 integration in AE350 is partitioned into 4 domains See Section 17.11.21.1. Each power domain is managed by a dedicated set of PCS (Power Control Slot) control registers.

17.11.1 Summary of Registers

SMU registers are summarized as follows:

Table 107: SMU Register Summary

Address Offset	Name	Description	Section
0x00	SYSTEMVER	SYSTEM ID & revision register	Section 17.11.2
0x08	SYSTEMCFG	SYSTEM configuration register	Section 17.11.3
0x0C	SMUVER	SMU version register	Section 17.11.4
0x10	WRSR	(*Legacy*) Wake-up and reset status register	Section 17.11.5
0x14	SMUCR	(*Legacy*) SMU command register	Section 17.11.6
0x1C	WRMASK	(*Legacy*) Wake-up and reset mask register	Section 17.11.7
0x20	CER	(*Legacy*) Clock enable register	Section 17.11.8
0x24	CRR	(*Legacy*) Clock ratio register	Section 17.11.9
0x40	SCRATCH	(*Legacy*) Scratch pad register	Section 17.11.10
0x44	HART_RESET_CTL	(*Legacy*) Hart reset control register	Section 17.11.11
0x50	RESET_VECTOR_LO	Hart reset vector register[31:0]	Section 17.11.12
0x60	RESET_VECTOR_HI	Hart reset vector register[63:32]	Section 17.11.13
(0x80+0x20*n)	PCS_CFG	Power control slot configuration register	Section 17.11.14
(0x84+0x20*n)	PCS_SCRATCH	Scratch pad	Section 17.11.15
(0x88+0x20*n)	PCS_MISC	Deep sleep mode setting and memory initial	Section 17.11.16
(0x8c+0x20*n)	PCS_MISC2	Partially clock control in light sleep mode	Section 17.11.17
(0x90+0x20*n)	PCS_WE	Power domain wakeup enable	Section 17.11.18
(0x94+0x20*n)	PCS_CTL	Power control slot control	Section 17.11.19
(0x98+0x20*n)	PCS_STATUS	Power control slot status	Section 17.11.20

17.11.2 SYSTEM ID & Revision Register (SYSTEMVER) (0x00)

Field Name	Bits	Description	Type	Reset
MINOR	[3:0]	Minor revision number	RO	0x0
MAJOR	[7:4]	Major revision number	RO	0x0
ID	[31:8]	ID for AE350	RO	0x414535

17.11.3 SYSTEM Configuration Register (SYSTEMCFG) (0x08)

Field Name	Bits	Description	Type	Reset
CORENUM	[7:0]	CPU core number	RO	0x1

17.11.4 SMU Version Register (SMUVER) (0x0c)

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>SMUVER</td><td>[31:0]</td><td>SMU Version<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0x0000</td><td>Legacy SMU</td></tr><tr><td>0x0100</td><td>ATCSMU100</td></tr></table>See Section 17.11.22.</td><td>RO</td><td>0x100</td></tr>
</table>

17.11.5 Wake-Up and Reset Status Register (WRSR) (0x10)

Legacy Usage for `SMUVER=0x0000`.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>APOR</td><td>[0]</td><td>AOPD Power-On Reset<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No action</td></tr><tr><td>1</td><td>Reset has occurred</td></tr></table></td><td>W1C</td><td>Note1</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>MPOR</td><td>[1]</td><td>MPD Power-On Reset
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No action</td></tr><tr><td>1</td><td>Reset has occurred</td></tr></table></td><td>W1C</td><td>Note2</td></tr>
<tr><td>HW</td><td>[2]</td><td>Hardware Reset
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Reset didn’t occur</td></tr><tr><td>1</td><td>Reset has occurred</td></tr></table></td><td>W1C</td><td>Note3</td></tr>
<tr><td>WDT</td><td>[3]</td><td>Watchdog Reset
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Reset didn’t occur</td></tr><tr><td>1</td><td>Reset has occurred</td></tr></table></td><td>W1C</td><td>Note3</td></tr>
<tr><td>SW</td><td>[4]</td><td>Software Reset
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Reset didn’t occur</td></tr><tr><td>1</td><td>Reset has occurred</td></tr></table></td><td>W1C</td><td>Note3</td></tr>
<tr><td>WI</td><td>[8]</td><td>Wake-up by external events
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Wake-up event didn’t occur</td></tr><tr><td>1</td><td>Wake-up event has occurred</td></tr></table></td><td>W1C</td><td>0</td></tr>
<tr><td>ALM</td><td>[9]</td><td>Wake-up by RTC alarm events
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Wake-up event didn’t occur</td></tr><tr><td>1</td><td>Wake-up event has occurred</td></tr></table></td><td>W1C</td><td>0</td></tr>
<tr><td>DBG</td><td>[10]</td><td>Wake-up by debug requests
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Wake-up event didn’t occur</td></tr><tr><td>1</td><td>Wake-up event has occurred</td></tr></table></td><td>W1C</td><td>0</td></tr>
</table>

Note

1. APOR is reset to 1 during the AOPD power-on reset.
2. MPOR is reset to 1 during the MPD power-on reset.
3. HW, WDT, and SW are reset to 0 during the AOPD power-on reset.

17.11.6 SMU Command Register (SMUCR) (0x14)

Legacy Usage for `SMUVER=0x0000`.

Field Name	Bits	Description	Type	Reset
SMUCMD	[7:0]	SMU command **Value** / **Meaning** 0x3c: Software reset to reset the whole system.	WO	0

17.11.7 Wake-Up and Reset Mask Register (WRMASK) (0x1c)

Legacy Usage for `SMUVER=0x0000`.

Field Name	Bits	Description	Type	Reset
WIMASK	[8]	Indicates whether external events will trigger wake-ups **Value** / **Meaning** 0: External events will trigger wake-up events 1: External events will not trigger wake-up events	RW	0

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>ALMMASK</td><td>[9]</td><td>Indicates whether RTC events will trigger wake-ups<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>RTC events will trigger wake-up events</td></tr><tr><td>1</td><td>RTC events will not trigger wake-up events</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>DBGMASK</td><td>[10]</td><td>Indicates whether debug requests will trigger wake-ups<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Debug requests will trigger wake-up events</td></tr><tr><td>1</td><td>Debug requests will not trigger wake-up events</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

17.11.8 Clock Enable Register (CER) (0x20)

Legacy Usage for `SMUVER=0x0000`.

This register controls all clocks in the platform. Processor and AHB/APB bus clocks should be turned on/off according to the programming sequence in Section 17.11.23.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CCLK_EN</td><td>[0]</td><td>Processor clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>HCLK_EN</td><td>[1]</td><td>AHB bus clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
</table>

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>PCLK_EN</td><td>[2]</td><td>Main APB bus clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>Reserved</td><td>[10:3]</td><td>Reserved</td><td>RO</td><td>0</td></tr>
<tr><td>ACLK_EN</td><td>[11]</td><td>AXI bus clock enable<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>0</td></tr>
</table>

17.11.9 Clock Ratio Register (CRR) (0x24)

Legacy Usage for `SMUVER=0x0000`.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CCLKSEL</td><td>[0]</td><td>Processor clock select<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>OSCH (Default)</td></tr><tr><td>1</td><td>Divide OSCH by 2</td></tr></table></td><td>RW</td><td>0</td></tr>
<tr><td>Reserved</td><td>[1]</td><td>Reserved</td><td>RO</td><td>0</td></tr>
<tr><td>HPCLKSEL</td><td>[4:2]</td><td>HCLK and PCLK clock ratio select<table><tr><th>Value</th><th>core_clk : aclk : hclk : pclk (frequency)</th></tr><tr><td>0</td><td>1:1:1:1</td></tr><tr><td>1</td><td>1:1:1:1/2</td></tr><tr><td>2</td><td>1:1:1:1/4</td></tr><tr><td>3</td><td>1:1:1/2:1/2</td></tr><tr><td>4</td><td>1:1:1/2:1/4</td></tr><tr><td>5-7</td><td>Reserved</td></tr></table></td><td>RW</td><td>IM</td></tr>
</table>

17.11.10 Scratch Pad Register (SCRATCH) (0x40)

Legacy Usage for `SMUVER=0x0000`.

This is a scratch register, which retains values when the rest of the system is powered down. It could be used to hold some parameters during the power down period.

Field Name	Bits	Description	Type	Reset
SCRATCH	[31:0]	Scratch register	RW	0

17.11.11 Hart Reset Control Register (HART_RESET_CTL) (0x44)

Legacy Usage for `SMUVER=0x0000`. The presence of this register is for the compatibility with the multi-core AE350 platform.

Field Name	Bits	Description	Type	Reset
HART0_RESET	[0]	Hardwired to 1	RO	1

17.11.12 Hart Reset Vector Register Low Part (RESET_VECTOR_LO) (0x50)

This register controls the value driven to the `reset_vector[31:0]` input signal of the NX25(F) processor.

Field Name	Bits	Description	Type	Reset
RESET_VECTOR	[31:0]	Entry address upon processor reset	RW	0x80000000

17.11.13 Hart Reset Vector Register High Part (RESET_VECTOR_HI) (0x60)

This register controls the value driven to the `reset_vector[63:32]` input signal of the NX25(F) processor.

Field Name	Bits	Description	Type	Reset
RESET_VECTOR	[31:0]	Entry address upon processor reset	RW	0x00000000

17.11.14 Power Control Slot Configuration Register (PCS_CFG) (0x80)

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Capability</td><td>[5:0]</td><td>Power control slot capability
<table>
<tr><th>Bit</th><th>Meaning</th></tr>
<tr><td>[0]</td><td>Reset</td></tr>
<tr><td>[1]</td><td>Reserved</td></tr>
<tr><td>[2]</td><td>Light Sleep</td></tr>
<tr><td>[3]</td><td>Deep Sleep</td></tr>
<tr><td>[4]</td><td>Reserved</td></tr>
<tr><td>[5]</td><td>Reserved</td></tr>
</table>
</td><td>RO</td><td>0x0</td></tr>
</table>

17.11.15 Scratch Pad (PCS_SCRATCH) (0x84)

This is a scratch register, which retains values when the rest of the system is powered down. It could be used to hold some parameters during the power down period.

Field Name	Bits	Description	Type	Reset
scratch	[31:0]	Scratch register	RW	0x0

17.11.16 Misc Register for Power Control Slot (PCS-MISC) (0x88)

This register is used to control the cycles for isolation/retention state changing and the memory reset initialization.

Field Name	Bits	Description	Type	Reset
iso_cyc	[3:0]	Cycles for isolation state changing	RW	0xf
ret_cyc	[11:4]	Cycles for retention state changing	RW	0xff
mem_init0	[28]	Memory 0 reset initialization	RW	1
mem_init1	[29]	Memory 1 reset initialization	RW	1
mem_init2	[30]	Memory 2 reset initialization	RW	1
mem_init3	[31]	Memory 3 reset initialization	RW	1

17.11.17 Misc Register 2 for Power Control Slot (PCS_MISC2) (0x8c)

This register is used to control the individual clock enables. See Section 17.11.22.1.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>CCLK_EN</td><td>[0]</td><td>Processor clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>HCLK_EN</td><td>[1]</td><td>AHB bus clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>PCLK_EN</td><td>[2]</td><td>Main APB bus clock enable.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table></td><td>RW</td><td>1</td></tr>
<tr><td>Reserved</td><td>[10:3]</td><td>Reserved</td><td>RO</td><td>0</td></tr>
<tr><td>ACLK_EN</td><td>[11]</td><td>AXI bus clock enable<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table>Available if SMUVER=0x0100 (ATCSMU100).</td><td>RW</td><td>1</td></tr>
<tr><td>LM_CLK_EN</td><td>[12]</td><td>Local memory clock enable<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table>Available if SMUVER=0x0100 (ATCSMU100).</td><td>RW</td><td>1</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>DC_CLK_EN</td><td>[13]</td><td>D-Cache clock enable
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disable clock</td></tr><tr><td>1</td><td>Enable clock</td></tr></table>Available if <code>SMUVER=0x0100 (ATCSMU100)</code>.
This bit is only valid in multi-core settings to keep D-Cache active to serve snooping traffic while the cores are in sleep modes. It is not used in single-core settings.</td><td>RW</td><td>1</td></tr>
</table>

17.11.18 Power Domain Wakeup Event Enable (PCS_WE) (0x90)

The enable registers for wakeup event. See Section 17.11.21.1 and Section 17.11.21.2.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>wakeup_cmd_en</td><td>[0]</td><td>This bit indicates the PCS wakeup command is supported, hardwired to 1</td><td>RO</td><td>0x1</td></tr>
<tr><td>wakeup_en</td><td>[31:1]</td><td>Each bit indicates one wakeup event
<table><tr><th>Bit[n]</th><th>Meaning</th></tr><tr><td>0</td><td>Disable corresponding wakeup event</td></tr><tr><td>1</td><td>Enable corresponding wakeup event</td></tr></table></td><td>RW</td><td>0x7fffffff</td></tr>
</table>

17.11.19 Power Control Slot Control Register (PCS_CTL) (0x94)

The control register for power control slot

Field Name	Bits	Description	Type	Reset
cmd	[2:0]	Power control command <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Active</td></tr><tr><td>1</td><td>PCS reset</td></tr><tr><td>2</td><td>Reserved</td></tr><tr><td>3</td><td>Sleep</td></tr><tr><td>4–7</td><td>Reserved</td></tr></table>	RW	0x0
param	[7:3]	The detail for power command When the cmd field is Active: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Reserved</td></tr><tr><td>1</td><td>Wakeup Command</td></tr></table>When the cmd field is PCS reset, this field means the minimal reset cycles. When the cmd field is sleep: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Light Sleep Mode</td></tr><tr><td>1</td><td>Deep Sleep Mode</td></tr></table>	RW	0x0

17.11.20 Power Control Slot Status Register (PCS_STATUS) (0x98)

The status register for power control slot

Field Name	Bits	Description	Type	Reset
pd_type	[2:0]	Power domain status <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Active</td></tr><tr><td>1</td><td>Reset</td></tr><tr><td>2</td><td>Sleep</td></tr><tr><td>3</td><td>Busy_Wait</td></tr></table>	RW	0x1

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>pd_status</td><td>[7:3]</td><td>The detail for power domain status
When pd_type is Active, pd_status indicates the wakeup reason:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Wakeup from PCS done</td></tr>
<tr><td>others</td><td>Wakeup from other events</td></tr>
</table>
When pd_type is Reset, pd_status indicates the reset reason:
<table>
<tr><th>Bit</th><th>Meaning</th></tr>
<tr><td>[3]</td><td>Power-on-reset in the always-on power domain</td></tr>
<tr><td>[4]</td><td>PCS_reset (SW reset)</td></tr>
<tr><td>[5]</td><td>Watch-dog reset</td></tr>
<tr><td>[6]</td><td>Hardware reset</td></tr>
<tr><td>[7]</td><td>Power-on-reset in the main power domain</td></tr>
</table>
When pd_type is Sleep, pd_status indicates the sleep mode type:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Light sleep</td></tr>
<tr><td>16</td><td>Deep sleep</td></tr>
<tr><td>others</td><td>Reserved</td></tr>
</table>
When pd_type is Busy_wait, pd_status indicates the wait reason:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Waiting on reset</td></tr>
<tr><td>2</td><td>Waiting on light sleep</td></tr>
<tr><td>3</td><td>Waiting on deep sleep</td></tr>
<tr><td>16</td><td>Busy on reset</td></tr>
<tr><td>18</td><td>Busy on light sleep</td></tr>
<tr><td>19</td><td>Busy on deep sleep</td></tr>
</table>
</td><td>RW</td><td>0x10</td></tr>
</table>

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>cmd_status</td><td>[10:8]</td><td>The detail for power command status
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Command successful</td></tr><tr><td>1–5</td><td>Reserved</td></tr><tr><td>6</td><td>Waive PCS command</td></tr><tr><td>7</td><td>Unsupported/Wrong PCS command</td></tr></table></td><td>RW</td><td>0x0</td></tr>
<tr><td>wakeup_int</td><td>[30]</td><td>Pending wakeup interrupt. This bit is asserted when PCS receives a wakeup event.</td><td>W1C</td><td>0x0</td></tr>
<tr><td>pend_int</td><td>[31]</td><td>Pending interrupt. This bit is asserted when the PCS command is invalid.</td><td>W1C</td><td>0x0</td></tr>
</table>

17.11.21 ATCSMU100 Integration in AE350

17.11.21.1 SMU Power Domain in AE350

The system is partitioned into 4 power domains as below:

Power Domain	Descriptions
PCS0	Always-on power domain, including the JTAG tap and DMI_AHB bus in NCEJDTM200.
PCS1	Power domain for the debug subsystem.
PCS2	Main power domain, including the system bus and AHB/APB peripheral IPs.
PCS3	Power domain for the processor core.

17.11.21.2 The Power Domain Wakeup Event

Each power domain is arranged to correlate to a set of wakeup events which are used to resume the domain. For example, SMU would wake up a powered-down domain upon receiving a timer interrupt event which is associated to this domain.

Most SMU wakeup events come from either harts or peripherals. The peripheral interrupts in Table 108 are connected to SMU as wakeup events.

Table 108: Peripheral Interrupt Sources for SMU Wakeup Events

Bits	Descriptions
[17]	Reserved
[16]	Reserved
[11]	BMC
[10]	DMA
[9]	UART2
[8]	UART1
[7]	GPIO
[6]	I2C
[5]	SPI2
[4]	SPI1
[3]	PIT
[2]	RTC alarm interrupt
[1]	RTC period interrupt

For PCS0 to PCS2 power domains, the signals in Table 109 are connected to the design as the wakeup events:

Table 109: The SMU Wakeup Event for PCS0–2

Bits	Descriptions
[31]	Hart0: meip/ueip/seip
[30]	Hart0: mtip
[29]	Hart0: msip
[28]	Hart0: debugint
[27:23]	Reserved
[22]	Hardware button (HW_RST_SW1) on evaluation board
[21]	dbg_wakeup_req
[20:1]	Peripheral interrupt source, see Table 108
[0]	Reserved

For PCS3 power domain, the signals in Table 110 are connected to the design as the wakeup events:

Table 110: The SMU Wakeup Event for PCS3

Bits	Descriptions
[31]	Hart0: meip/ueip/seip
[30]	Hart0: mtip
[29]	Hart0: msip
[28]	Hart0: debugint
[27]	Watch dog timer interrupt
[26:23]	Reserved
[22]	Hardware button (HW_RST_SW1) on evaluation board
[21]	dbg_wakeup_req
[20:1]	Peripheral interrupt source, see Table 108
[0]	Reserved

NMI is only included in PCS3 wakeup events. When the system hangs unexpectedly, PCS3 is expected to be resumed by NMI and resets the whole system.

17.11.22 SMU Programming Sequence

The registers after offset 0x80 are for PCS-based power control operations which provide fine-grain clock, reset and power control for each power domain. The registers 0x10–0x4F are for legacy power control operations, and they are for backward compatibility only. The software driver could detect the SMU version via `SMUVER` (0x0C) before doing further power operations.

Note

- The mixed use of PCS-based and legacy power control operations is not recommended.

17.11.22.1 ATCSMU100 Power Control Programming Sequence

The PCS-based power control operates under the following recommended software programming sequence and scenarios:

1. The software enables proper interrupts as wakeup events for power control operations.
2. The software issues the power operation via `PCS_CTL` and then executes the `WFI` instruction.
3. When the corresponding PCS in ATCSMU100 receives the command and captures the ready-to-execute condition (e.g., the processor enters WFI mode), ATCSMU100 performs power operation accordingly. The corresponding processor core is finally waken up by preset wakeup events.

The sample programming sequence for PCS Light Sleep operation:

1. Wait for or cancel the outstanding activities before issuing a PCS operation.
2. Set proper interrupts in PLIC and wakeup events in `PCS_WE`.
3. Issue the light sleep setting and command via `PCS_CTL` followed by a `WFI` instruction.
4. The clock of corresponding domain is turned off.

The sample programming sequence for PCS Deep Sleep operation:

1. Wait or cancel the outstanding activities before issuing a PCS operation.
2. Set proper interrupts in PLIC and wakeup events in `PCS_WE`.
3. Issue the deep sleep setting and command via `PCS_CTL` followed by a `WFI` instruction.
4. The power of corresponding domain is turned off.

17.11.23 Legacy SMU Programming Sequence

17.11.23.1 Legacy SMU Clock Control Flow

The legacy SMU supports simple clock control with the following programming sequence:

PROCESSOR CLOCK OPERATION SEQUENCE

1. Set RTC alarm in the RTC programming register (optional).
2. Set CCLK_EN to 0.
3. Set standby command in the SMU command register.
   - SMU issues an external interrupt to the processor notifying the standby request.
   - The processor should execute the `WFI` instruction to make the processor go into the WFI mode.
4. The Processor clock is disabled after the processor enters the WFI mode.
5. The Processor clock is enabled and the processor is waked up when a wake-up event arrives.
6. Clear the SMU command and status registers.

BUS CLOCK OPERATION SEQUENCE

1. Set RTC alarm in the RTC programming register (optional).
2. Set PCLK_EN, HCLK_EN or CCLK_EN to 0.
3. Set standby command in the SMU command register.
   - SMU issues an external interrupt to the processor notifying the standby request.
   - The processor should execute the `WFI` instruction to make the processor go into the WFI mode.
4. The bus clock or processor clock is disabled after the processor enters the WFI mode, depending on the PCLK_EN, HCLK_EN and CCLK_EN setting.
5. The bus clock is enabled and the processor is waked up when a wake-up event arrives.
6. Clear the SMU command and status registers.

17.11.24 The Wakeup Event Mask for Legacy SMU Usage

`WRMASK` is the wakeup event mask for legacy SMU. Table 111 shows the mask mapping from `WRMASK` to `PCS_WE` and the corresponding wakeup event bits are documented in Section 17.11.21.2.

Table 111: WRMASK to PCS_WE Mapping

WRMASK Fields	Value	PCS_WE Setting
WIMASK	0	PCS_WE[20:3] = 0x3ffff
	1	PCS_WE[20:3] = 0x0
ALMMASK	0	PCS_WE[2:1] = 0x3
	1	PCS_WE[2:1] = 0x0
DBGMASK	0	PCS_WE[28] = 0x1
	1	PCS_WE[28] = 0x0

17.11.25 Isolation Cell Emulation in FPGA

According to the power control flow, the WFI signal from the CPU core should be isolated at HIGH when the CPU core is powered off in the deep sleep mode. Since the isolation cell is not natively supported in FPGA, fake isolation cells are added during the FPGA synthesis. The fake isolation cells are implemented as a mux to select the isolation value in the deep sleep mode. The select signals come from SMU and go through a synchronizer from the `PCLK` domain to the `CORE_CLK` domain. Please note that the fake isolation cells and synchronizers are only present in the FPGA synthesis flow.

17.11.26 Simulation Model for Voltage and Power Control

The pd_vol_ctr module referenced inside the ae350_vol_ctrl module is a behavior model for voltage and power control under simulation. The model is expected to be replaced by a corresponding model or design in the real chip implementation.

The interface of the pd_vol_ctr module for each power slot is described in Table 112.

Table 112: Interface of ATCSMU100 to the Power Control Module

<table>
<tr><th>Signal Name</th><th>Direction</th><th>Description</th></tr>
<tr><td>pcs_vol_scale_req</td><td>output</td><td>The power control request signal. This request signal goes with <code>pcs_vol_scale</code> to request for voltage change.</td></tr>
<tr><td>pcs_vol_scale[2:0]</td><td>output</td><td>The voltage factor for DVFS. Only on/off is supported<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>power off</td></tr><tr><td>1</td><td>power on with voltage factor</td></tr></table></td></tr>
<tr><td>pcs_vol_scale_ack</td><td>input</td><td>Indicates the power/voltage change is done</td></tr>
</table>

Figure 20 is the example handshaking waveform for `pd_vol_ctr`. The handshaking would be:

- `pcs_vol_scale_req` is deasserted after `pcs_vol_scale_ack` is asserted.
- `pcs_vol_scale_ack` is deasserted after `pcs_vol_scale_req` is deasserted.

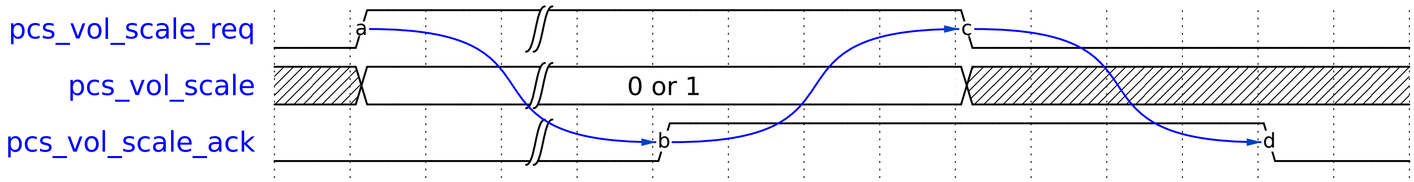


Figure 20: Handshaking of `pd_vol_ctr`

18 Platform-Level Interrupt Controller (PLIC)

18.1 Introduction

Andes Platform-Level Interrupt Controller (NCEPLIC100) prioritizes and distributes global interrupts. It is compatible with RISC-V PLIC with the following features:

- Configurable interrupt trigger types that are optionally programmable
- Software-programmable interrupt generation
- Preemptive priority interrupt extension
- Vectored interrupt extension

See Section 18.2 for information regarding the Andes preemptive priority interrupt extension and Section 18.3 for information regarding the Andes vectored PLIC extension.

The block diagram of NCEPLIC100 is shown in Figure 21. Interrupt sources (e.g., devices) send interrupt requests to NCEPLIC100 through `int_src` signals. The signals can be level-triggered or edge-triggered, and they are converted to interrupt requests by the interrupt gateway. Interrupt requests are prioritized and routed to interrupt targets (e.g., AndesCore processor cores) according to interrupt settings. Interrupt settings include enable bits, priorities, and priority thresholds, and these settings are programmable through the bus interface. Note that interrupt targets should not modify enable bits, priorities and priority thresholds if there are any un-serviced interrupts.

`t`*x*`_eip` (*x* stands for the target number) is an external interrupt pending notification signal to the targets. It is a level signal summarizing the interrupt pending (IP) status of all interrupt sources to target *x*. When a target takes the external interrupt, it should send an interrupt claim request (bus read request) to retrieve the interrupt ID, upon which the corresponding interrupt pending status bit will be cleared and `t`*x*`_eip` will be deasserted. `t`*x*`_eip` is guaranteed to be deasserted for at least one cycle even if there are pending interrupt sources still remaining. This is done to ensure that the interrupt detection logic of the target processor can see the remaining interrupt pending status.

The interrupt gateway stops processing newer interrupt requests from its interrupt sources once it reports an interrupt request. When the target has serviced the interrupt, it should send the interrupt completion message (bus write request) to NCEPLIC100 such that the interrupt gateway resumes processing newer interrupt requests.

The interrupt pending bit array of the PLIC registers provides a summary of all interrupt sources status. In addition, it is also writable for setting software-programmed interrupts for the corresponding interrupt sources. See Section 18.5.4 for more information.

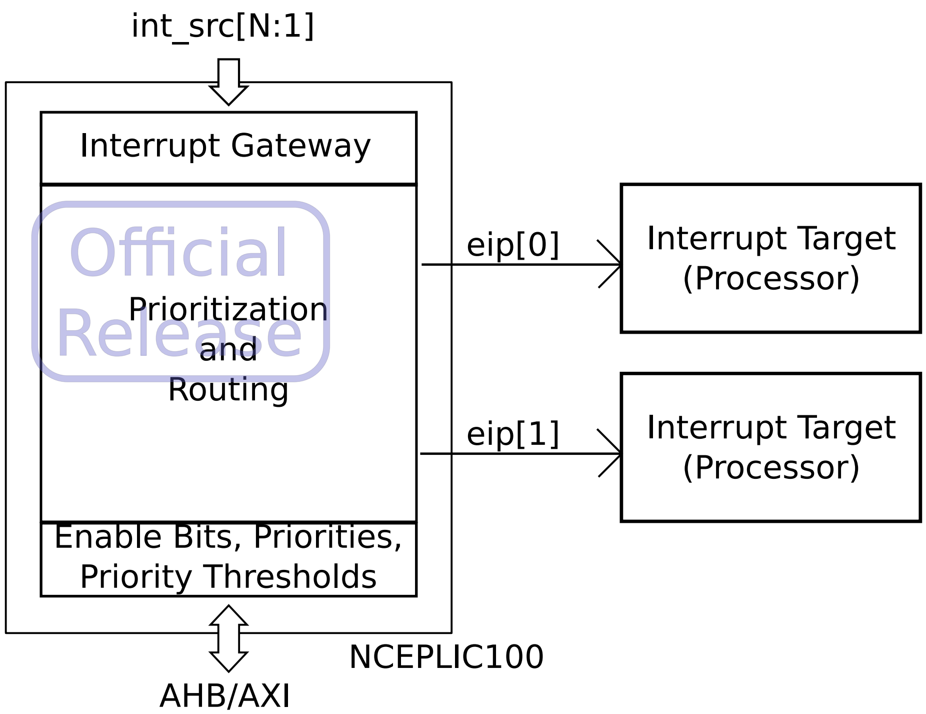


Figure 21: NCEPLIC100 Block Diagram

18.2 Support for Preemptive Priority Interrupt

NCEPLIC100 implements the Andes preemptive priority interrupt extension which enables faster responses for high-priority interrupts. This feature is enabled by setting the `PREEMPT` field (bit 0) of the Feature Enable Register (offset: 0x0000) to 1.

With this extension, if a high-priority interrupt arrives and the global interrupt is enabled (i.e.,`mstatus.MIE` is 1), the processor will stop servicing the current low-priority interrupt and begin servicing this new high-priority interrupt. The handling of the suspended lower-priority interrupts will resume only after the handling of the higher-priority interrupt ends. Interrupts of same or lower priorities will not cause preemption to take effect and interfere the handling of the current interrupt. They have to wait until the handling of the current interrupt finishes.

To support this feature, the PLIC core is enhanced with a preempted priority stack for each target. The stack saves and restores priorities of the nested/preempted interrupts for the target it is associated. The operation of the preempted stack is implicitly performed through two regular PLIC operations (*Interrupt Claim* and *Interrupt Completion*). See the next two subsections for more information.

18.2.1 Interrupt Claims with Preemptive Priority

When a target sends an interrupt claim message to the PLIC core, the PLIC core will atomically determine the ID of the highest-priority pending interrupt for the target and then deassert the corresponding source's IP bit. The PLIC core will then return the ID to the target.

At the same time, the priority number in the target's Priority Threshold Register will be saved to a preempted priority stack for that target and the new priority number of the claimed interrupt will be written to the Priority Threshold Register.

18.2.2 Interrupt Completion with Preemptive Priority

When a target sends an interrupt completion message to the PLIC core, in addition to forwarding the completion message to the associated gateway, the PLIC core will restore the highest priority number in the preempted priority stack back to the Priority Threshold Register of the target.

Note that out-of-order completion of interrupts is not allowed when this feature is turned on — the latest claimed interrupt should be completed first.

18.2.3 Programming Sequence to Allow Preemption of Interrupts

Turning on the global interrupt enable flag (`mstatus.MIE`) is all it takes to allow the current interrupt handler to be preempted by higher priority interrupts. However, as the preemptive priority stack operations do not allow out-of-order completion, some care should be taken to make sure that the claim and completion operations are nested properly.

For the non-vectored mode single-entry interrupt handler, the global interrupt enable flag could be turned on after the processor context are saved and *Interrupt Claim* is performed to allow preemption of the current interrupt handler. At the end of interrupt handler, an *Interrupt Completion* message is performed to signal that the handler has processed the interrupt and PLIC may deliver the next interrupt from the same interrupt source again. As both claim and completion messages are done through load/store instructions to device regions, they should automatically be ordered correctly. Compared with the vectored mode interrupt handler two paragraphs below, the global interrupt flag does not need to be disabled and no `FENCE` needs to be inserted after sending the completion message.

In summary, below is the suggested sequence for a non-vector mode interrupt handler for supporting preemptive priority interrupts:

1. Save registers/CSRs to stack
2. Sends *Interrupt Claim* message to PLIC (device-load)
3. Enable global interrupt (`mstatus.MIE`)

4. Handle the expected interrupt
5. Sends *Interrupt Completion* message to PLIC (device-store)
6. Restore registers/CSRs
7. Return from interrupt

For vector mode interrupt handlers (see the next section), *Interrupt Claim* is implicit when the external interrupt is taken. The global interrupt enable flag could be turned on as long as the processor context are saved to allow preemption of the current interrupt handler. However, the global interrupt flag should be turned off *before Interrupt Completion* operations are performed, since the processor will trigger the next implicit *Interrupt Claim* operation as soon as the global interrupt enable flag is turned on and cause races between *Interrupt Claim* and *Interrupt Completion*. Additionally, a `FENCE io,io` operation should be inserted after the *Interrupt Completion* operation to make sure that the completion message reaches PLIC before the interrupt handler returns, which turns on the interrupt enable flag again and cause the next *Interrupt Claim* to be performed.

In summary, below is the suggested sequence for a vector mode interrupt handler for supporting preemptive priority interrupts:

1. Save registers/CSRs to stack
2. Enable global interrupt (`mstatus.MIE`)
3. Handle the expected interrupt
4. Disable global interrupt (`mstatus.MIE`)
5. Sends *Interrupt Completion* message to PLIC (device-store)
6. Restore registers/CSRs
7. Use a `FENCE io,io` instruction to ensure that the completion message has reached PLIC.
8. Return from interrupt

18.3 Vectored Interrupts

NCEPLIC100 enhances the RISC-V PLIC functionality with the vector mode extension to allow the interrupt target to receive the interrupt source ID without going through the target claim request protocol. This feature can shorten the latency of interrupt handling by enabling the interrupt target to run the corresponding interrupt handler directly upon accepting the external interrupt. It is enabled by setting the `VECTORED` field of the Feature Enable Register (offset: 0x0000) to 1.

Two extra interface signals, `tx_eiid` and `tx_eiack`, are added to facilitate interrupt handling in the vector mode. When a valid interrupt is sent to PLIC, PLIC would send `tx_eip` with `tx_eiid`. Upon accepting the interrupt, the target asserts `tx_eiack` to PLIC and takes `tx_eiid` as the interrupt source ID. The assertion of `tx_eiack` would cause the deassertion of `tx_eip` and clearing of the `tx_IP` bit of corresponding interrupt source as does the handling of the interrupt claim request.

Note that interrupt *completion* messages are still required to notify the interrupt gateway the completion of interrupt handling and to forward additional interrupts to the PLIC core.

The interrupt priority arbitration works differently under the vector mode. In the non-vector mode, PLIC continues to arbitrate among all active interrupts even after the target is notified of occurrence of *some* interrupts (`tx_eip` sent to the target). Arbitration is not final until the claim request message is processed. In the vector mode, interrupt arbitration is final as soon as `tx_eip` is posted to the interrupt target. Interrupt arbitration resumes after `tx_eiack` is replied to PLIC and `tx_eip` is deasserted. The protocol does not change `tx_eiid` on the fly to allow PLIC and the interrupt target to operate at different clock domains.

The vector mode extension is designed such that each interrupt source is statically assigned to a single target. No two targets should compete servicing (claiming) the same interrupt source through the handshaking interface signals (`tx_eiack`). Otherwise, unpredictable results may occur.

Despite automatic dispatching, the PLIC interrupt claim request protocol still works under the vector mode for the interrupt handler to claim additional interrupts.

18.3.1 Vector Mode Protocol

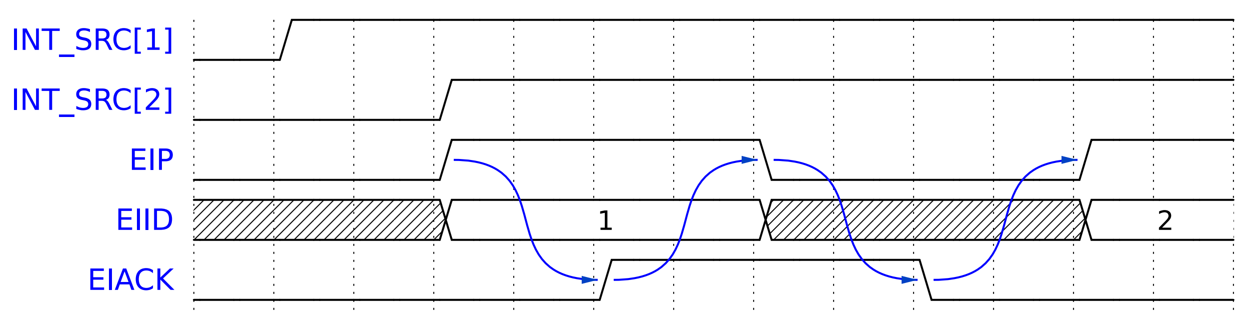


Figure 22: NCEPLIC100 Vector Mode Protocol

- `tx_eiid` remains stable when `tx_eip` is asserted.
- When `tx_eip` is asserted, it remains asserted until `tx_eiack` is asserted or an interrupt claim request is sent to PLIC.
- The assertion of `tx_eiack` causes the deassertion of `tx_eip`, which in turn causes the deassertion of `tx_eiack`.

- If there are more pending interrupts, `tx_eip` is asserted again after deassertion of `tx_eiack`.

18.4 PLIC Configuration Options

Table 113 summarizes all supported configuration parameters and the subsections describe the parameters in detail.

Table 113: PLIC Configuration Parameters

Parameter Name	Type	Valid Values	Default Value
INT_NUM	Integer	1–1023	63
TARGET_NUM	Integer	1–16	1
MAX_PRIORITY	Integer	3/7/15/31/63/127/255	15
PROGRAMMABLE_TRIGGER	Integer	See Section 18.4.4	0
EDGE_TRIGGER	Integer	See Section 18.4.5	0
ASYNC_INT	Integer	See Section 18.4.6	0
ADDR_WIDTH	Integer	≥22	32
DATA_WIDTH	Integer	32/64	32
VECTOR_PLIC_SUPPORT	String	yes/no	yes
PLIC_BUS	String	ahb/axi	axi
ID_WIDTH	Integer	4–32	4
SYNC_STAGE	Integer	2/3	2

18.4.1 Number of Interrupts

`INT_NUM` determines the number of interrupts, and the maximal value is 1023.

18.4.2 Number of Targets

`TARGET_NUM` determines the number of interrupt targets, and the maximal value is 16.

18.4.3 Maximum Interrupt Priority

`MAX_PRIORITY` determines the valid priority levels of the interrupt sources and the target threshold. The priority value 0 is reserved to mean “never interrupt”, and the larger the priority value, the higher the interrupt priority.

18.4.4 Programmable Trigger

PROGRAMMABLE_TRIGGER determines if the Interrupt Trigger Type registers are modifiable at run time.

- Value 0 means the trigger type registers are read-only.
- Value 1 means the trigger type registers are programmable at run time.

Trigger types are configured through the EDGE_TRIGGER parameter described in the next section. If they are programmable, the configured value will become their reset value.

18.4.5 Edge Trigger

EDGE_TRIGGER is regarded as a bit vector and each bit controls whether the corresponding interrupt source is level-triggered or edge-triggered:

- Value 0 means level-triggered; and
- Value 1 means edge-triggered.

The bit width of EDGE_TRIGGER should be (INT_NUM+1).

For example, value 0 means none of the interrupt source is edge-triggered.

18.4.6 Asynchronous Interrupt Source

ASYNC_INT is a bit vector where each bit indicates whether the corresponding interrupt source is asynchronous or synchronous.

- Value 0 means the interrupt source is synchronous.
- Value 1 means the interrupt source is asynchronous.

The bit width of ASYNC_INT should be (INT_NUM+1).

For example, value 0xc000000 means interrupt 26 and 27 of the interrupt source are asynchronous interrupts, and the rest are all synchronous ones.

18.4.7 Address Width of PLIC Bus Interface

ADDR_WIDTH determines the address width of PLIC bus. The address width should be greater than or equal to 22 to encompass all addressable PLIC memory space.

18.4.8 Data Width of PLIC Bus Interface

DATA_WIDTH determines the data width of PLIC bus.

18.4.9 Support For Vectored PLIC Extension

VECTOR_PLIC_SUPPORT controls whether to include Andes Vectored PLIC Extension or not. Note that while VECTOR_PLIC_SUPPORT is supported, harts can still perform *Interrupt Claim* operations through the AXI bus and the gate count difference of enabling VECTOR_PLIC_SUPPORT is minor to PLIC.

For PLIC integrated on the AE350 platform, VECTOR_PLIC_SUPPORT is automatically aligned to the CPU core setting by the configuration tool.

18.4.10 Bus Type of PLIC

PLIC_BUS determines the bus type of PLIC.

- String "ahb" indicates PLIC is an AHB slave device.
- String "axi" indicates PLIC is an AXI4 slave device.

18.4.11 ID Width of PLIC Bus Interface

ID_WIDTH determines the ID width of PLIC Bus Interface.

18.4.12 Synchronizer Level

SYNC_STAGE configures the level of the CDC synchronizer.

18.5 PLIC Registers

18.5.1 Summary of Registers

NCEPLIC100 registers are accessed through bus transfers, and the summary of registers is shown in Table 114.

Please note that NCEPLIC100 supports only 32-bit transfers. Behaviors of 8-bit and 16-bit transfers are UNDEFINED, and the transfers might be ignored or result in error responses.

Table 114: PLIC Register Summary

Address Offset		Description	Section
Begin	End		
0x000000	0x000003	Feature enable register	Section 18.5.2
0x000004	0x000007	Source 1 priority	Section 18.5.3
0x000008	0x00000B	Source 2 priority	
...	...	...	
0x000FFC	0x000FFF	Source 1023 priority	
0x001000	0x00107F	Pending array	Section 18.5.4
0x001080	0x0010FF	Trigger type array	Section 18.5.5
0x001100	0x001103	Number of interrupts and targets	Section 18.5.6
0x001104	0x001107	Version and max priority register	Section 18.5.7
0x002000	0x00207F	Target 0 interrupt enable bits	Section 18.5.8
0x002080	0x0020FF	Target 1 interrupt enable bits	
...	...	...	
0x002780	0x0027FF	Target 15 interrupt enable bits	
0x200000	0x200003	Target 0 priority threshold	Section 18.5.9
0x200004	0x200007	Target 0 claim/complete	Section 18.5.10
0x200400	0x20041F	Target 0 preempted priority stack	Section 18.5.11
0x201000	0x20141F	Target 1 priority threshold, claim/complete, preempted priority stack	
...	...	...	
0x20F000	0x20F41F	Target 15 priority threshold, claim/complete, preempted priority stack	

18.5.2 Feature Enable Register

Offset: 0x0

31–2	1	0
0	VECTORED	PREEMPT

This register enables preemptive priority interrupt feature and the vector mode.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>PREEMPT</td><td>[0]</td><td>Preemptive priority interrupt enable
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Disabled</td></tr>
<tr><td>1</td><td>Enabled</td></tr>
</table></td><td>RW</td><td>0</td></tr>
<tr><td>VECTORED</td><td>[1]</td><td>Vector mode enable
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Disabled</td></tr>
<tr><td>1</td><td>Enabled</td></tr>
</table>
Please note that both this bit and the `mmisc_ctl.VEC_PLIC` bit of the processor should be turned on for the vectored interrupt support to work correctly. See Section 15.10.7 for the definition of the `VEC_PLIC` bit.</td><td>RW</td><td>0</td></tr>
</table>

18.5.3 Interrupt Source Priority

Offset: n*4

31 ... 0
PRIORITY

This register determines the priority for interrupt source n ($1 \leq n \leq 1023$).

Field Name	Bits	Description	Type	Reset
PRIORITY	[31:0]	Interrupt source priority. The valid range of this field is determined by the MAX_PRIORITY field of the Version & Maximum Priority Configuration Register. **Value** / **Meaning** 0: Never interrupt. 1–255: Interrupt source priority. The larger the value, the higher the priority.	RW	1

18.5.4 Interrupt Pending

Offset: 0x1000 to 0x107F

These registers provide the interrupt pending status of interrupt sources, and a way for software to trigger an interrupt without relying on external devices. Every interrupt source occupies 1 bit. There are a total of 32 registers, each 32-bit wide, for 1023 interrupt sources. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

When these registers are read, the interrupt pending status of interrupt sources are returned. The pending bits could be set by writing a bit mask that specifies the bit positions to be set, and this action would result in software-programmed interrupts of the corresponding interrupt sources. The pending bits could only be cleared through the *Interrupt Claim* requests.

The location of the interrupt pending bit for interrupt source n ($1 \leq n \leq 1023$) can be determined by the following equations:

- Word offset address: 0x1000 + 4 * floor(n/32)
- Bit Position: n modulo 32

18.5.5 Interrupt Trigger Type

Offset: 0x1080 to 0x10FF

These registers indicate the interrupt trigger type of interrupt sources. Every interrupt source occupies 1 bit. There are a total of 32 registers, each 32-bit wide, for 1023 interrupt sources. The location of the interrupt trigger type bit for interrupt source n ($1 \leq n \leq 1023$) can be determined by the following equations:

- Word offset address: 0x1080 + 4 * floor(n/32)
- Bit Position: n modulo 32

The meaning of each bit is shown in Table 115. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

These registers are read only by default but can be optionally made programmable through the PROGRAMMABLE_TRIGGER configuration option.

Table 115: Meaning of Trigger Type

Value	Meaning
0	Level-triggered interrupt
1	Edge-triggered interrupt

18.5.6 Number of Interrupt and Target Configuration Register

Offset: 0x1100

31 16	15 0
NUM_TARGET	NUM_INTERRUPT

This register indicates the number of supported interrupt sources and supported targets.

Field Name	Bits	Description	Type	Reset
NUM_INTERRUPT	[15:0]	The number of supported interrupt sources	RO	IM
NUM_TARGET	[31:16]	The number of supported targets	RO	IM

18.5.7 Version & Maximum Priority Configuration Register

Offset: 0x1104

31 – 16	15 – 0
MAX_PRIORITY	VERSION

This register indicates the version and the maximum priority of PLIC implementation.

Field Name	Bits	Description	Type	Reset
VERSION	[15:0]	The version of the PLIC design	RO	IM
MAX_PRIORITY	[31:16]	The maximum priority supported	RO	IM

18.5.8 Interrupt Enable Bits for Target *m*

Offset: (0x2000+m*128) to (0x207F+m*128)

These registers control the routing of interrupt source n to target m ($1 \leq n \leq 1023$ and $m \geq 0$). Each bit controls one interrupt source. For each target, there are a total of 32 word-sized registers for controlling 1023 interrupt sources to that target. Note that zero is not a valid interrupt source number so bit 0 of the first register is hardwired to 0.

The location of the interrupt enable bit for interrupt source n to target m can be determined by the following equations:

- Word offset address: 0x2000 + 128 * m + 4 * floor(n/32)
- Bit position: n modulo 32

The following pseudo code demonstrates how to enable interrupt n for target m:

```
intptr_t reg_offset = 0x2000 + 128 * m + 4 * (n/32);
int bit_position = n % 32;

volatile uint32_t *reg_pointer = (volatile uint32_t *)(PLIC_BASE+reg_offset);

*reg_pointer = *reg_pointer | (1 << bit_position);
```

18.5.9 Priority Threshold for Target *m*

Offset: 0x200000+4096**m*

31 … 0
THRESHOLD

Each interrupt target m ($m \geq 0$) is associated with one Priority Threshold Register. Only active interrupts with priorities strictly greater than the threshold will cause an interrupt notification to be sent to the target.

Field Name	Bits	Description	Type	Reset
THRESHOLD	[31:0]	Interrupt priority threshold. The valid range of this field is determined by the MAX_PRIORITY field of the Version & Maximum Priority Configuration Register.	RW	0

18.5.10 Claim and Complete Register for Target *m*

Offset: 0x200004+4096*m

31 – 10	9 – 0
0	INTERRUPT_ID

There is one Claim and Complete Register for each interrupt target m ($m \geq 0$). Reading this register claims an interrupt source and returns the ID of that interrupt source.

The interrupt gateway stops processing newer interrupt requests from its interrupt sources until the earlier interrupt request completes. Writing this register with an interrupt ID serves as the interrupt completion message acknowledging to PLIC that the handling of the claimed interrupt has been serviced in target m and the associated interrupt gateway may resume processing newer interrupt requests.

The interrupt gateway only resumes processing of newer interrupt requests if the enable bit of the interrupt source for target m is set. If the enable bit is not set, the interrupt completion message will be ignored.

Generally there are no limitations to the order of interrupt claims and completions except when the preemptive priority mode is enabled. When PLIC is in the preemptive priority mode, the latest claimed interrupt should be completed first.

Field Name	Bits	Description	Type	Reset
INTERRUPT_ID	[9:0]	On reads, indicating the interrupt source that has being claimed. On writes, indicating the interrupt source that has been handled (completed).	RW	0

18.5.11 Preempted Priority Stack Registers for Target *m*

Offset: (0x200400+4096**m*) to (0x20041F+4096**m*)

31	30	29	28 … 3	2	1	0
PL031	PL030	PL029	...	PL002	PL001	PL000

31	30	29	28 … 3	2	1	0
PL063	PL062	PL061	...	PL034	PL033	PL032

...

31	30	29	28 … 3	2	1	0
PL255	PL254	PL253	...	PL226	PL225	PL224

These registers are read/writable registers for accessing the preempted priority stack for target m ($m \geq 0$). These registers are used for saving and restoring priorities of the nested/preempted interrupts for a particular target by hardware. They are made available to software primarily for diagnostic purposes.

There are a total of 8 registers, each 32-bit wide, for 255 priority levels. Each bit in these registers indicates if the corresponding priority level has been preempted by a higher-priority interrupt. The location of the priority level bit for priority p of target m (Word offset Address, Bit Position) can be determined by the following equations:

- Word offset Address: 0x20_0400 + 4096 * m + 4 * floor(p/32)
- Bit Position: p modulo 32

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>PL_p</td><td>[n]</td><td>This bit indicates that an interrupt at priority level p has been preempted by a higher-priority interrupt. The bit position n = p modulo 32.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No interrupts of priority level p are preempted.</td></tr>
<tr><td>1</td><td>An interrupt of priority level p has been preempted.</td></tr>
</table></td><td>RW</td><td>0</td></tr>
</table>

18.6 Interrupt Latency

Figure 23 illustrates the minimum timing for the processor to execute the first instruction.

When a device asserts INT_SRC[n], it takes 3 BUS_CLK cycles for NCEPLIC100 to arbitrate interrupts and assert its MEIP output signal, where BUS_CLK is the clock source of NCEPLIC100. When the MEIP signal is asserted, it takes 2 CORE_CLK cycles for the processor to latch the signal into the mip register.

How the processor fetches the trap handler for handling the associated external interrupt depends on its vector interrupt setting. When mmisc_ctl.VEC_PLIC is 0, the processor fetches the instruction pointed by mtvec. When mmisc_ctl.VEC_PLIC is 1 (vector mode), mtvec points to a table of entry point addresses, one entry for each external interrupt. It takes the processor 3 additional CORE_CLK cycles to fetch the entry point address, before fetching the first instruction of the associated trap handler. The waveform assumes that instruction fetch returns immediately without wait states.

The processor implements a 5-stage pipeline, so it takes at least 5 CORE_CLK cycles for the first instruction of the trap handler to execute and retire.

In summary, the minimum latency is 3 BUS_CLK and 8 CORE_CLK cycles. The latency is counted from assertion of INT_SRC[n] to the end of execution of the first instruction of the trap handler.

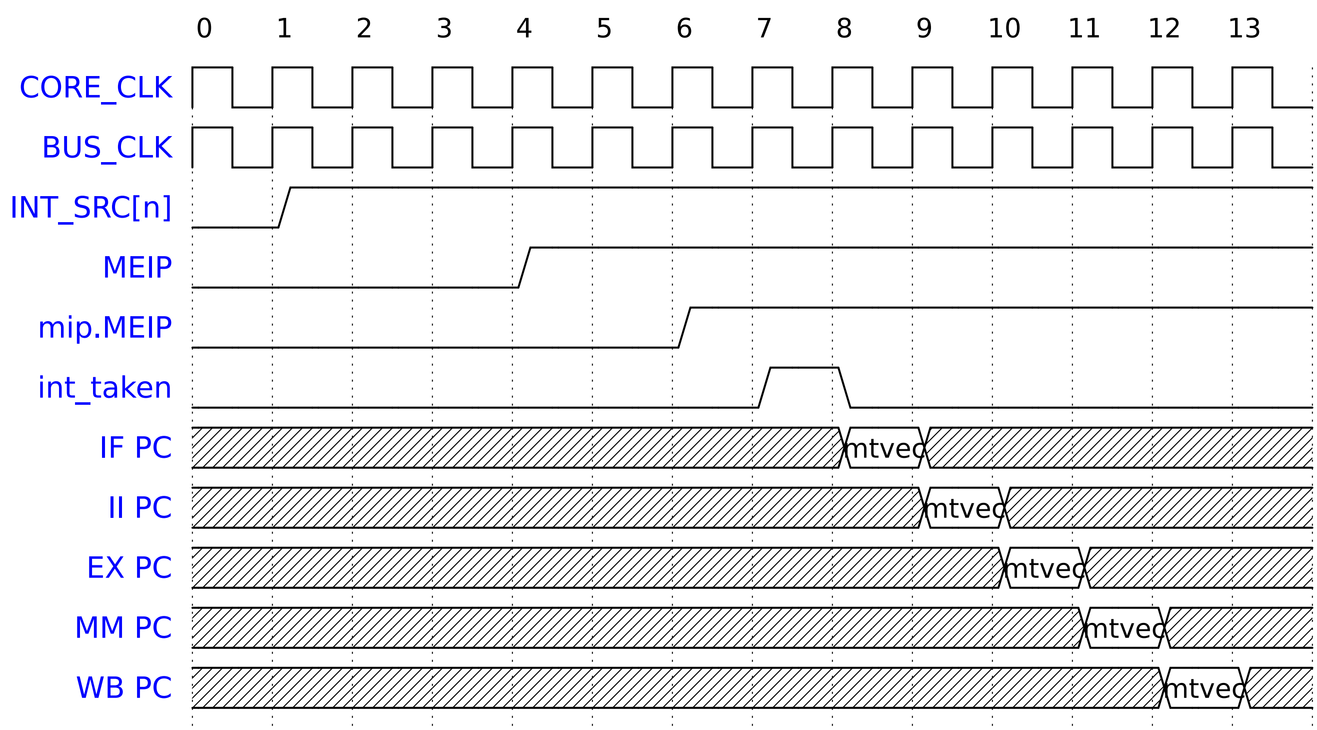


Figure 23: Minimum Interrupt Latency

18.7 Interface Signals

NCEPLIC100 offers two types of bus interfaces: AHB interface and AXI interface. The interfaces are selected by the PLIC_BUS parameter. The interface signals to both interfaces are simultaneously present on its module port list, and only the selected one will be used. The other group of signals will be unused and left floating.

The tables below describe the interface signals of NCEPLIC100, and the clock to NCEPLIC100 should be synchronous to that of NX25(F).

The valid transactions that NCEPLIC100 accepts are summarized in Table 119. It responds to invalid transactions by returning undefined values for read transactions and ignoring writes. On the AXI interface, rresp or bresp will also be set to SLVERR for invalid transactions.

Table 116: General Signals of NCEPLIC100

Signal Name	Direction	Description
clk	input	Clock
reset_n	input	Reset (Active-Low)
int_src[INT_NUM:1]	input	Interrupt sources. The sources could be configured to be asynchronous inputs through the ASYNC_INT parameter. See Section 18.4.6 for details.
t*x*_eip	output	External interrupt pending for target *x*.
t*x*_eiid[9:0]	output	External interrupt id for target *x*, see Section 18.3 for details.
t*x*_eiack	input	Interrupt acknowledgment from target *x*, see Section 18.3 for details.

Table 117: AHB Interface Signals of NCEPLIC100

Signal Name	Direction	Description
hsel	input	Slave Select
hrdata[31:0]	output	Read data bus
hready	input	Transfer done signal of AHB bus
hreadyout	output	Transfer done signal of PLIC
hresp[1:0]	output	Transfer response
haddr[ADDR_WIDTH-1:0]	input	Address bus
hburst[2:0]	input	Burst type
hprot[3:0]	input	Protection control

Continued on next page. . .

Table 117: (continued)

Signal Name	Direction	Description
hsize[2:0]	input	Transfer size
htrans[1:0]	input	Transfer type
hwdata[31:0]	input	Write data bus
hwrite	input	Transfer direction

Table 118: AXI Interface Signals of NCEPLIC100

Signal Name	Direction	Description
awid[3:0]	input	Write address ID
awaddr[ADDR_WIDTH-1:0]	input	Write address
awlen[7:0]	input	Write burst length
awsize[2:0]	input	Write burst size
awburst[1:0]	input	Write burst type
awlock	input	Write lock type
awcache[3:0]	input	Write cache type
awprot[2:0]	input	Write protection type
awvalid	input	Write address valid
awready	output	Write address ready
wdata[DATA_WIDTH-1:0]	input	Write data
wstrb[(DATA_WIDTH/8)-1:0]	input	Write strobes
wlast	input	Write last
wvalid	input	Write valid
wready	output	Write ready
bid[3:0]	output	Write response ID
bresp[1:0]	output	Write response
bvalid	output	Write response valid
bready	input	Write response ready
arid[3:0]	input	Read address ID
araddr[ADDR_WIDTH-1:0]	input	Read address
arlen[7:0]	input	Read burst length
arsize[2:0]	input	Read burst size
arburst[1:0]	input	Read burst type
arlock	input	Read lock type
arcache[3:0]	input	Read cache type

Continued on next page...

Table 118: (continued)

Signal Name	Direction	Description
arprot[2:0]	input	Read protection type
arvalid	input	Read address valid
arready	output	Read address ready
rid[3:0]	output	Read ID tag
rdata[DATA_WIDTH-1:0]	output	Read data
rresp[1:0]	output	Read response
rlast	output	Read last
rvalid	output	Read valid
rready	input	Read ready

Table 119: Valid Transactions for NCEPLIC100

Bus	Transaction Type
AHB	Single WORD, INCR WORD, WRAP4 WORD, INCR4 WORD, WRAP8 WORD, INCR8 WORD, WRAP16 WORD, INCR16 WORD
AXI	FIXED/INCR WORD with AxLEN=0

19 Machine Timer

19.1 Introduction

The RISC-V architecture defines a machine timer that provides a real-time counter and generates timer interrupts. NCEPLMT100 is an implementation of the machine timer, and the block diagram is shown in Figure 24. This timer is not to be confused with the real-time clock timer (RTC) of typical computing platforms. Per RISC-V privileged specification, the timer clock (`mtime_clk`) could be clocked at any arbitrary frequency as long as it is a fixed frequency clock that is not affected by clock gating or frequency scaling of clocks in the rest of the platform. On the other hand, RTC is usually clocked by a 32768Hz clock. The Linux kernel expects microsecond resolutions for `mtime`, so it imposes an additional requirement that the frequency of the timer clock has to be greater than 1MHz. For non-Linux applications, the timer clock could share the same clock source as the real-time clock timer.

The RISC-V privileged specification expects that software discovers the frequency of the timer clock through a platform specific mechanism. For Linux kernels, this is achieved through the Device Tree specification.

NCEPLMT100 imposes a frequency limitation on the frequency of the `mtime_clk` clock. The `mtime` update synchronization logic requires that the period of the `mtime_clk` should be more than twice that of the bus clock. Please take clock period variations into consideration when evaluating this constraint. Generally speaking, the slowest frequency of the bus interface of NCEPLMT100 should be more than 2x faster than that of the fastest `mtime_clk` clock.

On Andes evaluation platforms, the frequency of `mtime_clk` is set to be the same as the regular operating frequency of APB clocks to minimize the number of clock sources in FPGA.

NCEPLMT100 primarily consists of these memory-mapped registers: `mtime` and `mtimecmp`*n* (*n*: 0 – (NHART-1)). The `mtime` register is a 64-bit real-time counter clocked by `mtime_clk`.

The `mtimecmp`*n* register stores a 64-bit value for comparing with `mtime`. When the value in `mtime` is greater than or equal to the value in `mtimecmp`*n*, the `mtip[n]` signal is asserted for generating a timer interrupt. When `mtimecmp`*n* is written, the interrupt is cleared and the `mtip[n]` signal is deasserted.

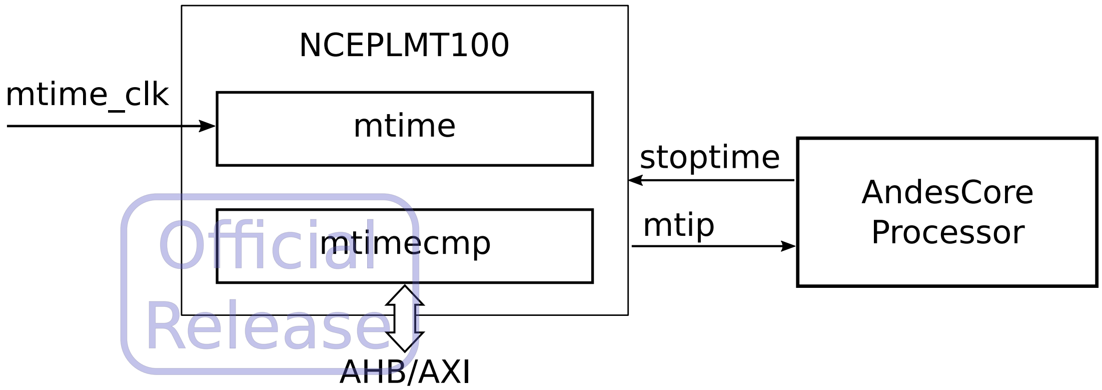


Figure 24: NCEPLMT100 Block Diagram

19.2 Machine Timer Registers

NCEPLMT100 registers are accessed through the bus interface, and their memory map is shown in Table 120.

Please note that NCEPLMT100 supports only 32-bit or 64-bit transfers. Behaviors of 8-bit and 16-bit transfers are UNDEFINED, and these transfers might be ignored as well as result in error responses or unexpected register updates.

Table 120: NX25(F) NCEPLMT100 Memory Map

Address Offset	Description
0x0 – 0x3	mtime[31:0]
0x4 – 0x7	mtime[63:32]
0x8 – 0xB	mtimecmp0[31:0]
0xC – 0xF	mtimecmp0[63:32]
0x10 – 0x13	mtimecmp1[31:0]
0x14 – 0x17	mtimecmp1[63:32]
0x18 – 0x1B	mtimecmp2[31:0]
0x1C – 0x1F	mtimecmp2[63:32]
0x8+*n**8 – 0xB+*n**8	mtimecmp*n*[31:0]
0xC+*n**8 – 0xF+*n**8	mtimecmp*n*[63:32]

The `mtime` register is driven by `mtime_clk`, which is assumed to be slower than the clock of the bus interface. The `mtime_shadow` shadow register is maintained in the bus clock domain to reduce the latency of accessing the `mtime` register in the slow clock domain. The values of `mtime` and `mtime_shadow` registers are constantly synchronized such that `mtime_shadow` maintains the most up-to-date values of `mtime`. The value in `mtime_shadow` is instantly returned when reading the `mtime` register. When writing the `mtime` register, bus write transactions finish when the values are written to the `mtime_shadow` register, and NCEPLMT100 handles the synchronization to `mtime` in the background.

19.2.1 Machine Timer Initialization

The `mtime` counter is a 64-bit value and it increments non-stop on every machine timer clock except the first few cycles after its control register updates. When the data bus of NCEPLMT100 is 64-bit, it is recommended to program the `mtime` counter through a single double-word store to guarantee the value is updated atomically. Otherwise, the following programming sequence should be followed to initialize the counter through two separate 32-bit updates to `mtime[31:0]` and `mtime[63:32]`.

- If bit[31:29] of the intended value is 7:
  1. Write *zero* to `mtime[31:0]`.
  2. Write high part of the intended value to `mtime[63:32]`.
  3. Write low part of the intended value to `mtime[31:0]`.
- Otherwise:
  1. Write low part of the intended value to `mtime[31:0]`.
  2. Write high part of the intended value to `mtime[63:32]`.

19.3 Machine Timer Configuration Options

Table 121 summarizes all supported configuration parameters and the subsections describe the parameters in detail.

Table 121: NCEPLMT100 Configuration Parameters

Parameter Name	Type	Valid Values	Default Value
ADDR_WIDTH	Integer	$\geq$10	32
DATA_WIDTH	Integer	32 or 64	32
BUS_TYPE	String	ahb/axi	ahb
ID_WIDTH	Integer	$\geq$1	4
NHART	Integer	1–32	4
SYNC_STAGE	Integer	2 or 3	2

19.3.1 Address Width

ADDR_WIDTH configures the address width of Machine Timer bus. The address width should be greater than or equal to 10 to encompass all addressable Machine timer memory space.

19.3.2 Data Width

DATA_WIDTH configures the data width of Machine Timer bus. It is either 32-bit or 64-bit.

19.3.3 Number of Supported Harts

NHART configures the number of supported harts. This essentially configures the number of timer comparison registers `mtimecmp`N as well as the width of `mtip` interface, where N = 0 – (NHART-1).

19.3.4 Bus Type

BUS_TYPE configures the bus type of Machine Timer.

- String “ahb” indicates Machine Timer is an AHB slave device.
- String “axi” indicates Machine Timer is an AXI4 slave device.

19.3.5 AXI ID Width

ID_WIDTH configures the width of the AXI ID signals. This parameter only takes effect when BUS_TYPE is “axi”.

19.3.6 Synchronizer Level

SYNC_STAGE configures the level of the CDC synchronizer.

19.4 Interface Signals

NCEPLMT100 offers two types of bus interfaces: AHB and AXI interfaces. The interfaces are selected by the `PLMT_BUS` parameter. The interface signals to both interfaces are simultaneously present on its module port list, and only the selected one will be used. The other group of signals will be unused and left floating.

The tables below describe the interface signals of NCEPLMT100, and the clock to NCEPLMT100 should be synchronous to that of NX25(F).

The valid transactions that NCEPLMT100 accepts are summarized in Table 125.

It responds to invalid transactions by returning undefined values for read transactions and ignoring writes. On the AXI interface, non-zero `rresp` or `bresp` is also set to `SLVERR` for invalid transactions.

Table 122: General Signals of NCEPLMT100

Signal Name	Direction	Description
clk	input	Clock for the bus interface
resetn	input	Reset for the bus interface (Active-Low)
mtime_clk	input	Clock for the `mtime` counter. See Section 19.1
por_rstn	input	Power on reset (Active-Low) for initializing the `mtime` counter
stoptime	input	Stop counting the `mtime` counter
mtip[NHART-1:0]	output	Timer interrupt pending

Table 123: AHB Interface Signals of NCEPLMT100

Signal Name	Direction	Description
hsel	input	Slave select
hrdata[DATA_WIDTH-1:0]	output	Read data bus
hready	input	Transfer done signal of the AHB bus
hreadyout	output	Transfer done signal of Machine Timer
hresp[1:0]	output	Transfer response
haddr[ADDR_WIDTH-1:0]	input	Address bus
hburst[2:0]	input	Burst type
hprot[3:0]	input	Protection control
hsize[2:0]	input	Transfer size
htrans[1:0]	input	Transfer type
hwdata[DATA_WIDTH-1:0]	input	Write data bus
hwrite	input	Transfer direction

Table 124: AXI Interface Signals of NCEPLMT100

Signal Name	Direction	Description
awid[3:0]	input	Write address ID

Continued on next page. . .

Table 124: (continued)

Signal Name	Direction	Description
awaddr[ADDR_WIDTH-1:0]	input	Write address
awlen[7:0]	input	Write burst length
awsize[2:0]	input	Write burst size
awburst[1:0]	input	Write burst type
awlock	input	Write lock type
awcache[3:0]	input	Write cache type
awprot[2:0]	input	Write protection type
awvalid	input	Write address valid
awready	output	Write address ready
wdata[DATA_WIDTH-1:0]	input	Write data
wstrb[(DATA_WIDTH/8)-1:0]	input	Write strobes
wlast	input	Write last
wvalid	input	Write valid
wready	output	Write ready
bid[3:0]	output	Write response ID
bresp[1:0]	output	Write response
bvalid	output	Write response valid
bready	input	Write response ready
arid[3:0]	input	Read address ID
araddr[ADDR_WIDTH-1:0]	input	Read address
arlen[7:0]	input	Read burst length
arsize[2:0]	input	Read burst size
arburst[1:0]	input	Read burst type
arlock	input	Read lock type
arcache[3:0]	input	Read cache type
arprot[2:0]	input	Read protection type
arvalid	input	Read address valid
arready	output	Read address ready
rid[3:0]	output	Read ID tag
rdata[DATA_WIDTH-1:0]	output	Read data
rresp[1:0]	output	Read response
rlast	output	Read last
rvalid	output	Read valid

Continued on next page...

Table 124: (continued)

Signal Name	Direction	Description
rready	input	Read ready

Table 125: Valid Transactions for NCEPLMT100

Bus	Configuration	Transaction Type
AHB	DATA_WIDTH == 32	Single WORD
AHB	DATA_WIDTH == 64	Single WORD, Double WORD
AXI	DATA_WIDTH == 32	Single WORD
AXI	DATA_WIDTH == 64	Single WORD, Double WORD

20 Debug Subsystem

20.1 Overview

The debug subsystem implements *RISC-V External Debug Support (TD003) V0.13*. Figure 25 shows the block diagram of the debug subsystem, which contains two components: NCEPLDM200 and NCEJDTM200. NCEPLDM200 is the debug module, which could be accessed through its two AHB slave ports. One is the system interface port, which is for an AndesCore processor to access the debug module through the system bus. The other one is the Debug Memory Interface (DMI) port, which is accessed by NCEJDTM200 (JTAG Debug Transport Module). NCEJDTM200 converts debug commands in JTAG interfaces of external debuggers to bus read/write requests to the DMI port.

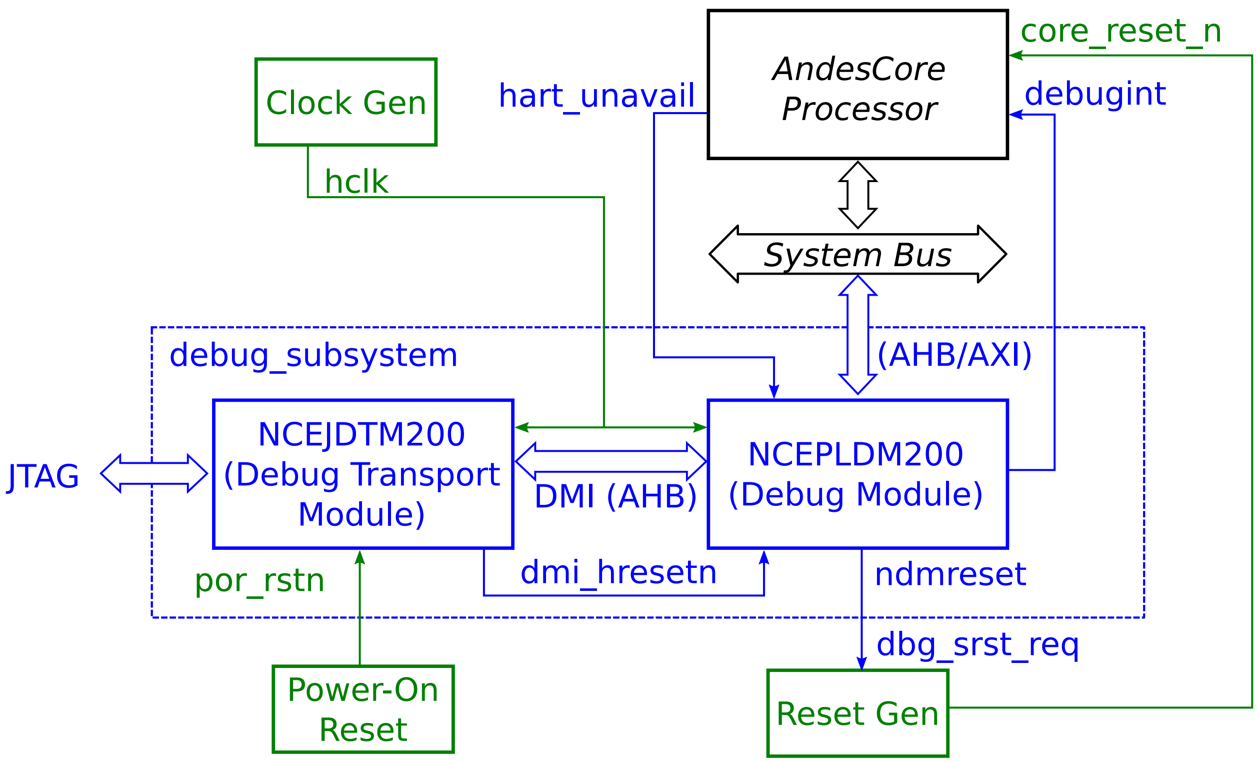


Figure 25: Debug Subsystem Block Diagram

Debug interrupts cause the processor to enter the debug mode and redirect the instruction fetch to the debug exception handler, whose entry point should be the base address of NCEPLDM200 and defined by the Debug Module Base option of NX25(F).

NCEPLDM200 includes a Debug ROM at its base address that defines the debug exception handler. When invoked, the debug handler polls NCEPLDM200 internal registers to process commands issued by the external debugger through the NCEJDTM200 module. Typical debug commands include accessing processor registers, accessing memories, and executing programs written in the Program Buffer, which is a memory region writable by the external debugger.

Note

As debug commands of the external debugger are serviced through the exception handler in Debug Rom, it is a design limitation that the debug facility may not be accessible if the system bus hangs. The system bus matrix can have a timeout mechanism to complete timed-out transactions if the system freezes to facilitate debugging on hardware failures.

Correct operations of the debug subsystem require clocks, resets and I/O interfaces to be connected in a certain way, as well as making sure Debug Module Base is within a device region (Section 2.12). Please see the next subsection for information.

20.2 Integration Requirements

For proper operations of the debug subsystem, the following platform-level requirements of reset, clock and I/O signals should be met:

- NCEJDTM200 should be reset by power-on resets, which will not be triggered by other resets in the system.
- NCEPLDM200 should be reset by `dmi_hresetn` that is generated by NCEJDTM200.
- `ndmreset` (non-debug-module reset) should connect to the platform reset generator, which triggers platform-wide resets. Through the reset generator, `ndmreset` should be able to control `core_reset_n` of AndesCore processors. As the name suggests, `ndmreset` should reset neither NCEJDTM200 nor NCEPLDM200. Please also note that the platform reset should not affect the pin-muxing of the external debug interface (JTAG) pins to preserve connectivity to the external debugger throughout resets.
- `ndmreset` should not cause assertion of power-on resets (`pwr_rst_n`).
- The clock signal (`clk`) to NCEJDTM200 and NCEPLDM200 should keep running during the assertion of `ndmreset`. Stopped clocks will impede the deassertion of `ndmreset`.
- The bus interfaces of NCEJDTM200 and NCEPLDM200 should be in the same clock domain.
- Both `tck` and `tms` pins of the JTAG interface should not be floating when the external debugger is not attached:

- tck can be either pulled up to HIGH or pulled down to LOW.
- tms should be pulled up to HIGH.

20.3 Optional Debug Subsystem

By default, the debug subsystem is embedded in the platform.

For the AE350 platform, the debug subsystem follows the core debug setting (Section 2.9.1). The debug subsystem can be removed by simply disabling the core debug feature in the configuration tool.

20.4 Debug Subsystem Configuration Options

Table 126: Debug Subsystem Configuration Parameters

Parameter Name	Type	Valid Values	Default Value
NHART	Integer	1–1024	1
SYS_ADDR_WIDTH	Integer	9–64	32
SYS_DATA_WIDTH	Integer	32, 64, or 128	64
ADDR_WIDTH	Integer	9–64	32
DATA_WIDTH	Integer	32 or 64	64
SYS_BUS_ACCESS	String	“yes” or “no”	yes*
DEBUG_INTERFACE	String	“jtag” or “serial”	jtag*
PROGBUF_SIZE	Integer	1–8	8*
HALTGROUP_COUNT	Integer	0–31	0
RESUMEGROUP_SUPPORT	String	“yes” or “no”	no
DMXTRIGGER_COUNT	Integer	0–16	0

Note

- These options should be configured in
  - configuration tool (Section 2.10, for AE350).

20.4.1 Number of Harts

NHART determines how many harts can be connected to a NCEPLDM200 module. The maximum value is 1024.

20.4.2 Bus Slave Configurations

ADDR_WIDTH and DATA_WIDTH determine the address width and the data width of the bus slave interface. This interface is used for debug flow control and data exchange with all the connected harts. All the ports on this interface use "rv" as prefix.

20.4.3 System Bus Access

SYS_BUS_ACCESS determines whether the optional feature, system bus access is supported. With system bus access support, PLDM can access the memory without involving any hart.

20.4.4 System Bus Master Configurations

SYS_ADDR_WIDTH and SYS_DATA_WIDTH determine the address width and the data width of the bus master. This interface is only used for system bus access feature. All the ports on this interface use "sys" as prefix.

20.4.5 Debug Interface

DEBUG_INTERFACE determines the interface between debug subsystem and the external device for debug.

- String "jtag" implies the standard 4-wire JTAG interface.
- String "serial" implies Andes 2-wire serial debug interface.

20.4.6 Program Buffer Size

PROGBUF_SIZE is used to configure the size of program buffer, in 32-bit words.

20.4.7 Halt Group Configuration

Halt group is an optional feature that allows debug module to halt all the harts when any of the hart in the same halt group is halted. HALTGROUP_COUNT determines how many halt groups are supported besides the default halt group (group #0). No halt groups are configured when HALTGROUP_COUNT is zero.

20.4.8 Resume Group Configuration

Resume Group is an optional feature that allows debug module to resume all the harts when any of the hart in the same resume group is resumed. RESUMEGROUP_SUPPORT selects whether this feature is supported. The number of resume groups will be equal to the number of halt groups and is configured by HALTGROUP_COUNT of the Halt Group feature. It should not be zero when this feature is configured.

20.4.9 External Triggers

External triggers are interface signals to signal halt/resume activity from another debug domain. DMXTRIGGER_COUNT determines how many external triggers are configured. A value of 0 means no external triggers are configured. Each configured external trigger consists of a halt in/out pair (xtrigger_halt_in/out) and a resume in/out pair (xtrigger_resume_in/out).

This feature requires the Halt Group feature to be configured. The resume group feature will be implicitly enabled as well when this feature is configured.

The input signals serve as requests to trigger the group halt/resume actions and the output signals serve as acknowledge signals reflecting the group event getting triggered for the respective input signal.

All external triggers are assigned to group 0 by default, which means they do not participate in any group halt/resume actions and no cross trigger action will be performed. An external trigger needs to be programmed to join halt/resume groups to trigger group halt/resume events in the debug module. Please see dmcs2 for details.

Note

- The RISC-V debug specification defines two kinds of external triggers: Debug Module External Trigger and Trigger Module External Trigger. External Trigger here is the Debug Module External Trigger, not to be confused with the Trigger Module External Trigger.

20.5 NCEPLDM200

Table 127 summarizes the memory map within the NCEPLDM200 address space as viewed from the system bus interface. Please note that the offset for the program buffer could be discovered by the external debugger through execution of the `AUIPC` instruction as the first instruction in the program buffer, and the starting offset of Abstract Data is defined as hartinfo.DATAADDR. The offsets could be used as offsets of load/store instructions with the `zero` register as the base register to access this memory space. The `zero` register is automatically mapped to the base of NCEPLDM200 for load/store instructions in Debug Mode.

This system bus address space of NCEPLDM200 is defined through the Debug Module Base option. It should be within a device region for the proper operation of the Debug ROM and the external debugger support. Please see Section 2.12 for information on how to set up this address space as device regions.

Table 128 summarizes the memory map as viewed from the DMI interface. The address[8:2] value in the table follows the address value assignment of the debug module debug bus registers as described in *RISC-V External Debug Support (TD003) V0.13*.

Table 127: System Memory Map of NCEPLDM200

Address Offset	Description	Definition
0x0000 – 0x007F	Debug ROM	Internal Use Only
0x0080 – 0x00BF	Program Buffer	Section 20.5.13
0x00C0 – 0x00CF	Abstract Data 0–3	Section 20.5.1
0x00D0 – 0x01FF	Reserved for internal use	N/A

Table 128: DMI Memory Map of NCEPLDM200

Address[8:2]	Description	Definition
0x04 – 0x07	Abstract Data 0–3	Section 20.5.1
0x10	Debug Module Control	Section 20.5.2
0x11	Debug Module Status	Section 20.5.3
0x12	Hart Info	Section 20.5.4
0x13	Halt Summary 1	Section 20.5.6
0x14	Hart Array Window Select	Section 20.5.7
0x15	Hart Array Window	Section 20.5.8
0x16	Abstract Control and Status	Section 20.5.9
0x17	Abstract Command	Section 20.5.10
0x18	Abstract Command Autoexec	Section 20.5.11

Continued on next page. . .

Table 128: (continued)

Address[8:2]	Description	Definition
0x19 – 0x1C	Device Tree Addr 0–3	Section 20.5.12
0x20 – 0x2F	Program Buffer 0–15	Section 20.5.13
0x30	Authentication Data	Section 20.5.14
0x32	Debug Module Control and Status 2	Section 20.5.15
0x38	System Bus Access Control and Status	Section 20.5.16
0x39 – 0x3B	System Bus Address	Section 20.5.17
0x3C – 0x3F	System Bus Data	Section 20.5.18
0x40	Halt Summary 0	Section 20.5.5

20.5.1 Abstract Data 0–3 (`data0` – `data3`)

Basic read/write registers that may be read or changed by abstract commands.

The registers are accessible from both the DMI interface and the system bus interface to support data exchanges between the external debugger and the processor (i.e., instructions in the program buffer).

20.5.2 Debug Module Control (`dmcontrol`)

15–4	3	2	1	0
0	setresethaltreq	clrresethaltreq	ndmreset	dmactive

31	30	29	28	27	26	25–16
haltreq	resumereq	0	ackhavereset	0	hasel	hartsel

Field Name	Bits	Description	Type	Reset
dmactive	[0]	Controlling reset signal for Debug Module itself. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The Debug Module's state takes its reset values.</td></tr><tr><td>1</td><td>The Debug Module functions normally.</td></tr></table>	RW	0x0
ndmreset	[1]	Controlling reset signal from the Debug Module to the rest of the system. <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Deassert system reset signal.</td></tr><tr><td>1</td><td>Assert system reset signal.</td></tr></table>	RW	0x0
clrresethaltreq	[2]	This optional field clears the halt-on-reset request bit for all currently selected harts. Writes apply to the new value of `hartsel` and `hasel`.	W1	-

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>setresethaltreq</td><td>[3]</td><td>This optional field writes the halt-on-reset request bit for all currently selected harts, unless clrresethaltreq is simultaneously set to 1. When set to 1, each selected hart will halt upon the next deassertion of its reset. The halt-on-reset request bit is not automatically cleared. The debugger must write to clrresethaltreq to clear it. Writes apply to the new value of hartsel and hasel. If hasresethaltreq is 0, this field is not implemented.</td><td>W1</td><td>-</td></tr>
<tr><td>hartsel</td><td>[25:16]</td><td>Selecting the target hart to be debugged.</td><td>RW</td><td>0x0</td></tr>
<tr><td>hasel</td><td>[26]</td><td>Selects the definition of currently selected harts.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>There is a single currently selected hart, that is selected by hartsel.</td></tr>
<tr><td>1</td><td>There may be multiple currently selected harts selected by hartsel, plus those selected by the hart array mask register.</td></tr>
</table>
</td><td>RW</td><td>0x0</td></tr>
<tr><td>ackhavereset</td><td>[28]</td><td>Writing 1 to this bit clears the havereset bits for any selected harts. Harts are selected based on the new value of hartsel and hasel being written.</td><td>W1</td><td>0x0</td></tr>
<tr><td>resumereq</td><td>[30]</td><td>Writes the resume request bit for all currently selected harts. When set to 1, each selected hart will resume if it is currently halted. The resume request bit is ignored while the halt request bit is set. Harts are selected based on the new value of hartsel and hasel being written.</td><td>WO</td><td>0x0</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
haltreq	[31]	Writes the halt request bit for all currently selected harts. When set to 1, each selected hart will halt if it is not currently halted. Writing 1 or 0 has no effect on a hart which is already halted, but the bit must be cleared to 0 before the hart is resumed. Harts are selected based on the new value of `hartsel` and `hasel` being written.	WO	0x0

Note

- NCEPLDM200 uses wraparound hart ID to select harts, which can reduce the integration effort on a multi-hart system with multiple debug modules. When a hart ID is larger than the effective bit value, it will be mapped to a smaller value. For example, hart 10 will be remapped to hart 2 on a debug module instance supporting 8 harts (NHART=8). Essentially, the effective bits of hart ID used in NCEPLDM200 is the ceiling of log2 value of `NHART`.

20.5.3 Debug Module Status (`dmstatus`)

9	8	7	6	5	4	3 0
allhalted	anyhalted	authenticated	authbusy	hasresethaltreq	devtreevalid	version

15	14	13	12	11	10
allnonexistent	anynonexistent	allunavail	anyunavail	allrunning	anyrunning

31 23	22	21 20	19	18	17	16
0	impebreak	0	allhavereset	anyhavereset	allresumeack	anyresumeack

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>version</td><td>[3:0]</td><td>Version of the implemented RISC-V External Debug Support. 0x2 indicates that the current implemented version is 0.13.</td><td>RO</td><td>0x2</td></tr>
<tr><td>devtreevalid</td><td>[4]</td><td>Whether the information in devtreeaddr0 – devtreeaddr3 registers holds the address of the Device Tree.</td><td>RO</td><td>0x0</td></tr>
<tr><td>hasresethaltreq</td><td>[5]</td><td>This bit indicates the supported status of <code>resethaltreq</code>.</td><td>RO</td><td>0x1</td></tr>
<tr><td>authbusy</td><td>[6]</td><td><table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The authentication module is ready to process the next read/write to authdata.</td></tr><tr><td>1</td><td>The authentication module is busy.</td></tr></table></td><td>RO</td><td>0x0</td></tr>
<tr><td>authenticated</td><td>[7]</td><td><table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Authentication is required before using the Debug Module.</td></tr><tr><td>1</td><td>Authentication check has passed.</td></tr></table></td><td>RO</td><td>0x1</td></tr>
<tr><td>anyhalted</td><td>[8]</td><td>Indicates whether any currently selected hart is halted.</td><td>RO</td><td>0x0*</td></tr>
<tr><td>allhalted</td><td>[9]</td><td>Indicates whether all currently selected harts are halted.</td><td>RO</td><td>0x0*</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
anyrunning	[10]	Indicates whether any currently selected hart is running.	RO	0x1*
allrunning	[11]	Indicates whether all currently selected harts are running.	RO	0x1*
anyunavail	[12]	Indicates whether any currently selected hart is unavailable.	RO	0x0*
allunavail	[13]	Indicates whether all currently selected harts are unavailable.	RO	0x0*
anynonexistent	[14]	Indicates whether any currently selected hart does not exist in this system.	RO	0x0*
allnonexistent	[15]	Indicates whether all currently selected harts do not exist in this system.	RO	0x0*
anyresumeack	[16]	Indicates whether any currently selected hart has acknowledged the previous resume request.	RO	0x0*
allresumeack	[17]	Indicates whether all currently selected harts have acknowledged the previous resume request.	RO	0x0*
anyhavereset	[18]	Indicates whether any currently selected hart has been reset but the reset has not been acknowledged.	RO	0x0*
allhavereset	[19]	Indicates whether all currently selected harts have been reset but the reset has not been acknowledged.	RO	0x0*
impebreak	[22]	Indicates whether there is an implicit `EBREAK` instruction at the nonexistent word immediately after the program buffer.	RO	0x1

Note

- The reset values with * mark may not reflect the transient hart states when NCEPLDM200 is under reset or `dmcontrol.DMACTIVE` is not set.

20.5.4 Hart Info (`hartinfo`)

31–24	23–20	19–17	16	15–12	11–0
0	nscratch	0	dataaccess	datasize	dataaddr

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>dataaddr</td><td>[11:0]</td><td>Signed offset for accessing the shadowed data registers by the processor, to be used as offsets for load/store instructions with the <code>zero</code> register as the base register in Debug Mode.</td><td>RO</td><td>0xC0</td></tr>
<tr><td>datasize</td><td>[15:12]</td><td>Number of 32-bit words in the memory map dedicated to shadowing the data registers.</td><td>RO</td><td>0x4</td></tr>
<tr><td>dataaccess</td><td>[16]</td><td>The method for accessing the shadowed data registers. The value of this field is 0x1 for the processor, indicating that the data registers are shadowed in the memory map of the selected hart under Debug Mode.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The data registers are shadowed in the hart by CSR registers.</td></tr>
<tr><td>1</td><td>The data registers are shadowed in the hart's memory map. Each register takes up 4 bytes in the memory map.</td></tr>
</table></td><td>RO</td><td>0x1</td></tr>
<tr><td>nscratch</td><td>[23:20]</td><td>Number of dscratch registers available for the debugger to use during program buffer execution, starting from <code>dscratch0</code>.</td><td>RO</td><td>0x2</td></tr>
</table>

20.5.5 Halt Summary 0 (`haltsum0`)

31 … 0
haltsum0

Each bit of this register indicates whether one specific hart is halted or not. Unavailable/nonexistent harts are not considered to be halted.

The LSB reflects the halt status of hart `(hartsel[9:5]<<5)`, and the MSB reflects the halt status of `(\(hartsel[9:5]<<5) + 31)`.

This register is read-only.

20.5.6 Halt Summary 1 (`haltsum1`)

31 … 0
haltsum1

Each bit of this register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted.

The LSB reflects the halt status of harts 0x0 through 0x1f. The MSB reflects the halt status of harts 0x3e0 through 0x3ff.

This register is read-only.

20.5.7 Hart Array Window Select (`hawindowsel`)

31 … 15	14 … 0
0	hawindowsel

Field Name	Bits	Description	Type	Reset
hawindowsel	[14:0]	This register selects which of the 32-bit portion of the hart array mask register is accessible in hawindow.	RW	0x0

20.5.8 Hart Array Window (`hawindow`)

31 … 0
hawindow

Field Name	Bits	Description	Type	Reset
hawindow	[31:0]	This register provides R/W accesses to a 32-bit portion of the hart array mask register. The position of the window is determined by `hawindowsel`. That is, bit 0 refers to hart (`hawindowsel` * 32), while bit 31 refers to hart (`hawindowsel` * 32 + 31).	RW	0x0

20.5.9 Abstract Control and Status (`abstractcs`)

31 29	28 24	23 13	12	11	10 8	7 5	4 0
0	progbufsize	0	busy	0	cmderr	0	datacount

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>datacount</td><td>[4:0]</td><td>Number of data registers that are implemented.</td><td>RO</td><td>0x4</td></tr>
<tr><td>cmderr</td><td>[10:8]</td><td>Error code indicating that an abstract command fails. The bits in this field remain set until they are cleared by writing 1 to them. No abstract command is started until the value is reset to 0.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>None: no error</td></tr>
<tr><td>1</td><td>Busy: an abstract command was executing while command, abstractcs, or abstractauto was written, or when one of the data or progbuf registers was read or written.</td></tr>
<tr><td>2</td><td>Not supported: the requested command is not supported.</td></tr>
<tr><td>3</td><td>Exception: an exception occurred while executing the command.</td></tr>
<tr><td>4</td><td>Halt/resume: an abstract command cannot execute because the hart is not in the expected state (running/halted).</td></tr>
</table></td><td>R/W1C</td><td>0x0</td></tr>
<tr><td>busy</td><td>[12]</td><td>Flag indicating an abstract command is currently being executed.</td><td>RO</td><td>0x0</td></tr>
<tr><td>progbufsize</td><td>[28:24]</td><td>Size of the program buffer, in 32-bit words.</td><td>RO</td><td>Configuration Dependent</td></tr>
</table>

20.5.10 Abstract Command (command)

31–24	23–0
cmdtype	control

Field Name	Bits	Description	Type	Reset
control	[23:0]	The field is interpreted in a command-specific manner.	WO	0x0
cmdtype	[31:24]	Controlling the overall functionality of this abstract command.	WO	0x0

If a command takes arguments or returns values, they must be written to or placed in the data registers. Which data registers are used for the arguments is described in Table 129.

Table 129: Use of Data Registers in PLDM

Argument Width	arg0/Return Value	arg1	arg2
32	data0	data1	data2
64	data0, data1	data2, data3	data4, data5

20.5.10.1 Access Register

31–24	23	22–20	19	18	17	16	15–0
cmdtype (0)	0	size	0	postexec	transfer	write	regno

This command gives the debugger access to CPU registers and allows CPU to execute the program buffer.

Field Name	Bits	Description	Type	Reset
regno	[15:0]	Index of the specified register to be accessed.	WO	0x0

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>write</td><td>[16]</td><td>The direction of data transfer.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Copy data from the specified register into arg0 portion of the Abstract Data registers.</td></tr><tr><td>1</td><td>Copy data from arg0 portion of Abstract Data registers into the specified register.</td></tr></table></td><td>WO</td><td>0x0</td></tr>
<tr><td>transfer</td><td>[17]</td><td>Indicates whether to perform data transfer.<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Do not do the operation specified by write.</td></tr><tr><td>1</td><td>Do the operation specified by write.</td></tr></table></td><td>WO</td><td>0x0</td></tr>
<tr><td>postexec</td><td>[18]</td><td>Indicates whether to execute the program in the program buffer.
If this field is set, execute the program in the Program Buffer exactly once after performing the transfer, if any.</td><td>WO</td><td>0x0</td></tr>
<tr><td>size</td><td>[22:20]</td><td><table><tr><th>Value</th><th>Meaning</th></tr><tr><td>2</td><td>Access the lowest 32 bits of the register</td></tr><tr><td>3</td><td>Access the lowest 64 bits of the register.</td></tr></table></td><td>WO</td><td>0x0</td></tr>
</table>

20.5.10.2 Quick Access

31 ... 24	23 ... 0
cmdtype (1)	0

Perform the following sequence of operations:

- If the hart is halted already, the command sets cmderr to *halt/resume* and does not continue.

- Halt the hart. If the hart halts for some other reason (e.g., breakpoint), the command sets cmderr to *halt/resume* and does not continue.
- Execute the program buffer. If an exception occurs, cmderr is set to exception and the program buffer execution ends, but the quick access command continues.
- Resume the hart.

20.5.10.3 Access Memory

31–24	23	22–20	19	18–17	16	15–0
cmdtype (2)	aamvirtual	aamsize	aampostincrement	0	write	0

This command lets the debugger perform memory accesses with the memory view of the selected hart.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>write</td><td>[16]</td><td>The direction of data transfer.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Copy data from the memory location specified in arg1 into arg0 portion of data.</td></tr>
<tr><td>1</td><td>Copy data from arg0 portion of data into the memory location specified in arg1.</td></tr>
</table></td><td>WO</td><td>0x0</td></tr>
<tr><td>aampostincrement</td><td>[19]</td><td>Increment arg1 by the number of bytes encoded in <code>aamsize</code> after a memory access completes.</td><td>WO</td><td>0x0</td></tr>
<tr><td>aamsize</td><td>[22:20]</td><td>Size of memory accesses.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Access the lowest 8 bits of the memory location.</td></tr>
<tr><td>1</td><td>Access the lowest 16 bits of the memory location.</td></tr>
<tr><td>2</td><td>Access the lowest 32 bits of the memory location.</td></tr>
<tr><td>3</td><td>Access the lowest 64 bits of the memory location.</td></tr>
</table></td><td>WO</td><td>0x0</td></tr>
</table>

Continued on next page...

Field Name	Bits	Description	Type	Reset
aamvirtual	[23]	Virtual or physical address accesses **Value** / **Meaning** 0 / Addresses are physical 1 / No action	WO	0x0

20.5.11 Abstract Command Autoexec (`abstractauto`)

31–24	23–16	15–12	11–4	3–0
0	autoexecprogbuf	0	0	autoexecdata

Field Name	Bits	Description	Type	Reset
autoexecdata	[3:0]	When a bit in this field is 1, read or write accesses to the corresponding data word cause the command in command to be executed again.	RW	0x0
autoexecprogbuf	[23:16]	When a bit in this field is 1, read or write accesses to the corresponding progbuf word cause the command in command to be executed again.	RW	0x0

20.5.12 Device Tree Addr 0–3 (`devtreeaddr0` – `devtreeaddr3`)

The *devicetreeaddr* registers are hardwired to zeros in NCEPLDM200.

20.5.13 Program Buffer 0–15 (`progbuf0` – `progbuf15`)

31–0
data

The *progbuf* registers provide read/write accesses to the program buffer. NCEPLDM200 supports only first *abstractcs.progbufsize* entries, the other entries are hardwired to 0x00100073 (the `EBREAK` instruction).

These registers are read/write accessible from both the DMI interface and the system bus, in addition to being a valid region for instruction fetches. They hold small programs written by the external debugger and these small programs will be fetched and executed by the processor upon execution of the abstract commands which require execution of the program buffer.

Programs in *progbuf* must end with `EBREAK` or `C.EBREAK` instructions if the program size is less than *abstractcs.progbufsize* words (32 bytes).

20.5.14 Authentication Data (`authdata`)

The *authdata* register is hardwired to zeros in NCEPLDM200.

20.5.15 Debug Module Control and Status 2 (`dmcs2`)

31–12	11	10–7	6–2	1	0
0	grouptype	dmexttrigger	group	hgwrite	hgselect

The *dmcs2* register contains DM control and status bits that do not easily fit in *dmcontrol* and *dmstatus*. Currently, *dmcs2* only contains the control and status bits for group halt/resume. See Section 20.5.22 for more information.

Field Name	Bits	Description	Type	Reset
hgselect	[0]	This bit indicates whether external triggers or the selected harts are controlled by *dmcs2*. Value: 0 — Operate on harts Value: 1 — Operate on external triggers	WARL	0x0

Continued on next page. . .

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>hgwrite</td><td>[1]</td><td>Writing 1 to this field assigns halt/resume group to selected harts or external triggers.
• When hgselect is 0, update the halt/resume group of all selected harts to the value written to group.
• When hgselect is 1, update the halt/resume group of the selected external trigger to the value written to group.
Writing 0 reads out the halt/resume group of the selected hart/external trigger.
The grouptype field selects halt or resume group to update.</td><td>WO</td><td>-</td></tr>
<tr><td>group</td><td>[6:2]</td><td>This field contains the halt/resume group of the target selected by hgselect.
• When hgselect is 0, this field contains the halt/resume group of the hart specified by hartsel.
• When hgselect is 1, this field contains the halt/resume group of the external trigger selected by dmexttrigger.
The grouptype field selects whether this field contains a halt group number or a resume group number.</td><td>WARL</td><td>0x0</td></tr>
<tr><td>dmexttrigger</td><td>[10:7]</td><td>This field contains the currently selected external trigger. If no external trigger exists, this field will be hardwired to 0.</td><td>WARL</td><td>0x0</td></tr>
<tr><td>grouptype</td><td>[11]</td><td>This field controls whether the rest of the fields in this register applies to halt groups or resume groups.
<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Halt groups</td></tr><tr><td>1</td><td>Resume groups</td></tr></table></td><td>WARL</td><td>0x0</td></tr>
</table>

20.5.16 System Bus Access Control and Status (sbcs)

15	14 12	11 5	4	3	2	1	0
sbreadondata	sberror	sbasize	sbaccess128	sbaccess64	sbaccess32	sbaccess16	sbaccess8

31 29	28 23	22	21	20	19 17	16
sbversion	0	sbbusyerror	sbbusy	sbreadonaddr	sbaccess	sbautoincrement

The *sbcs* register holds the control bits and status flags of System Bus Accesses. It is only valid when the SYSTEM_BUS_ACCESS_SUPPORT parameter of NCEPLDM200 is set. It is otherwise hardwired to zero.

Field Name	Bits	Description	Type	Reset
sbaccess8	[0]	This bit indicates the supported status of 8-bit system bus data accesses.	RO	0x1
sbaccess16	[1]	This bit indicates the supported status of 16-bit system bus data accesses.	RO	0x1
sbaccess32	[2]	This bit indicates the supported status of 32-bit system bus data accesses.	RO	0x1
sbaccess64	[3]	This bit indicates the supported status of 64-bit system bus data accesses. This bit is 1 if SYS_DATA_WIDTH == 64.	RO	Configuration Dependent
sbaccess128	[4]	This bit indicates the supported status of 128-bit system bus data accesses.	RO	0x0
sbasize	[11:5]	Address width of System bus addresses in bits. This field is 0 if there is no bus access support. Otherwise, it is the value of the SYS_ADDR_WIDTH parameter of NCEPLDM200.	RO	Configuration Dependent

Continued on next page...

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>sberror</td><td>[14:12]</td><td>Error code indicating the failure type of the system bus accesses. Write 1 to clear the status.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No bus error</td></tr>
<tr><td>1</td><td>Timeout error</td></tr>
<tr><td>2</td><td>Bad address</td></tr>
<tr><td>3</td><td>Alignment error</td></tr>
<tr><td>4</td><td>Unsupported size was requested</td></tr>
<tr><td>7</td><td>Others</td></tr>
</table></td><td>R/W1C</td><td>0x0</td></tr>
<tr><td>sbreadondata</td><td>[15]</td><td>When 1, Every read from sbdata0 automatically triggers a system bus read at the (possibly auto-incremented) address.</td><td>RW</td><td>0x0</td></tr>
<tr><td>sbautoincrement</td><td>[16]</td><td>sbaddress is incremented by the size specified in sbaccess_ after every system bus access.</td><td>RW</td><td>0x0</td></tr>
<tr><td>sbaccess</td><td>[19:17]</td><td>Access size.
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>8-bit</td></tr>
<tr><td>1</td><td>16-bit</td></tr>
<tr><td>2</td><td>32-bit</td></tr>
<tr><td>3</td><td>64-bit</td></tr>
<tr><td>4</td><td>128-bit</td></tr>
</table></td><td>RW</td><td>0x2</td></tr>
<tr><td>sbreadonaddr</td><td>[20]</td><td>When 1, Every write to sbaddress0 automatically triggers a system bus read at the new address.</td><td>RW</td><td>0x0</td></tr>
<tr><td>sbbusy</td><td>[21]</td><td>Indicate the system bus master is busy.</td><td>RO</td><td>0x0</td></tr>
<tr><td>sbbusyerror</td><td>[22]</td><td>Indicate the debugger attempts to execute system bus access before sbbusy is cleared.</td><td>R/W1C</td><td>0x0</td></tr>
<tr><td>sbversion</td><td>[31:29]</td><td>The version of the supported System Bus Access version. It is currently 1 for v0.13 of the RISC-V External Debug Support specification.</td><td>RO</td><td>0x1</td></tr>
</table>

Note

The states of *sbaccess8* – *sbaccess128* are set based on the `SYS_ADDR_WIDTH` parameter in NCE-PLDM200.

20.5.17 System Bus Address (`sbaddress0` – `sbaddress2`)

The *sbaddress* registers are only valid when the `SYSTEM_BUS_ACCESS_SUPPORT` parameter of NCE-PLDM200 is set. They are otherwise hardwired to zeros.

Table 130: System Bus Address Register

Address Register	Description
sbaddress0	bit[31:0] of the address in *sbaddress*
sbaddress1	bit[63:32] of the address in *sbaddress*
sbaddress2	bit[95:64] of the address in *sbaddress*
sbaddress3	bit[127:96] of the address in *sbaddress*

20.5.18 System Bus Data (`sbdata0` – `sbdata3`)

The *sbdata* register is supported as below when `SYSTEM_BUS_ACCESS_SUPPORT` parameter of NCE-PLDM200 is set. Otherwise it is hardwired to zeros.

Table 131: System Bus Data Register

Address Register	Description
sbdata0	bit [31:0] of the address in *sbdata*
sbdata1	bit [63:32] of the address in *sbdata*
sbdata2	bit [95:64] of the address in *sbdata*
sbdata3	bit [127:96] of the address in *sbdata*

20.5.19 Interface Signals

The interface signals consist of four groups of interfaces: a DMI interface, a bus slave interface, a system bus master interface and the General Signals interface.

The DMI interface is a dedicated bus interface for communicating with debug transport modules (NCEJDTM200).

The bus slave interface is the bus interface for the target processor and the Debug Module to exchange data.

The system bus master interface is an optional interface for supporting the direct System Bus Access feature of the RISC-V External Debug Specification. This feature is not enabled by default and it can be turned on through the SYSTEM_BUS_ACCESS_SUPPORT parameter of NCEPLDM200. The interface signals, however, are always present on the module port list regardless of configuration. They should be left floating and ignored when not enabled.

NCEPLDM200 offers two types of bus interfaces for the bus slave interface and the system bus interface: AHB interface and AXI interface. The interface types are selected by the RV_BUS_TYPE and SYS_BUS_TYPE parameters. The interface signals of both types are simultaneously present on the module port list and only the selected one will be used. The other group of signals will be unused and left floating. The values of RV_BUS_TYPE and SYS_BUS_TYPE parameters should match the bus type of the processor. Note that the bus protocol of the DMI interface is not configurable and unconditionally uses the AHB interface.

The General Signals include the clock and reset signals, processor status and debug-interrupt signals. NCEPLDM200 supports multi-hart (multi-target) debugging. The number of harts supported is controlled by its NHART parameter, and the width of all hart-specific signals are NHART-wide, one bit per hart.

The tables below describe the interface signals of NCEPLDM200, and the clock to NCEPLDM200 should be synchronous to that of the processor. All signals are Active-High unless otherwise indicated.

Table 132: General Signals of NCEPLDM200

Signal Name	Direction	Description
clk	input	Clock input. This clock should also drive the DMI interface of NCEJDTM200. Furthermore, this clock should not be stopped during ndmreset. See Section 20.2 for integration requirements.
reset_n	input	Reset (Active-Low). The reset signal should be driven by NCEJDTM200. It should not be active when ndmreset is asserted. See Section 20.2 for integration requirements.
hart_unavail[NHART-1:0]	input	Hart unavailable, one bit per hart. Each bit will be 1 if the corresponding hart is not available for accesses by the external debugger. The hart could be in the reset or some kind of power-gating state.

Continued on next page...

Table 132: (continued)

Signal Name	Direction	Description
hart_under_reset[NHART-1:0]	input	Hart under reset, one bit per hart. Each bit will be 1 if the corresponding hart is under reset.
debugint[NHART-1:0]	output	Debug interrupt, one bit per hart. Each bit will be 1 if the external debugger makes a debug request to the corresponding hart. Each hart should respond to the request by entering the debug mode.
dmactive	output	Debug module active. This signal reflects the value of dmcontrol.DMACTIVE.
ndmreset	output	Non-debug module reset. This signal should be routed to the platform reset controller, so that the external debugger could trigger system reset for the platform. See Section 20.2 for integration requirements.
resethaltreq[NHART-1:0]	output	Halt-on-reset request, one bit per hart. Each bit will be 1 if the external debugger sets the corresponding hart to enter the debug mode after reset (the halt-on-reset requests).

Table 133: DMI Interface Signals of NCEPLDM200

Signal Name	Direction	Description
dmi_hrdata[31:0]	output	DMI read data bus
dmi_hreadyout	output	DMI transfer done of NCEPLDM200
dmi_hresp[1:0]	output	DMI transfer response
dmi_hsel	input	DMI selection
dmi_htrans[1:0]	input	DMI transfer type
dmi_haddr[9:0]	input	DMI address bus
dmi_hburst[2:0]	input	DMI burst type
dmi_hprot[3:0]	input	DMI protection control
dmi_hsize[2:0]	input	DMI transfer size
dmi_hready	input	DMI transfer done
dmi_hwrite	input	DMI transfer direction
dmi_hwdata[31:0]	input	DMI write data bus

Table 134: AHB Slave Signals of NCEPLDM200

Signal Name	Direction	Description
rv_hrdata[DATA_WIDTH-1:0]	output	Processor read data bus
rv_hreadyout	output	Processor transfer done of NCEPLDM200
rv_hresp[1:0]	output	Processor transfer response
rv_haddr[ADDR_WIDTH-1:0]	input	Processor address bus
rv_hburst[2:0]	input	Processor burst type
rv_hprot[3:0]	input	Processor protection control
rv_hsize[2:0]	input	Processor transfer size
rv_htrans[1:0]	input	Processor transfer type
rv_hwdata[DATA_WIDTH-1:0]	input	Processor write data bus
rv_hwrite	input	Processor transfer direction
rv_hsel	input	Processor selection
rv_hready	input	Processor transfer done

Table 135: RV AHB Slave Transactions Acceptable by NCEPLDM200

Configuration	Transaction Type
DATA_WIDTH == 32	SINGLE WORD
DATA_WIDTH == 64	SINGLE WORD
	SINGLE DOUBLEWORD

Table 136: AHB Master Signals of NCEPLDM200

Signal Name	Direction	Description
sys_hrdata[SYS_DATA_WIDTH-1:0]	input	System bus read data bus
sys_hready	input	System bus transfer done
sys_hgrant	input	System bus bus grant
sys_hresp[1:0]	input	System bus transfer response
sys_haddr[SYS_ADDR_WIDTH-1:0]	output	System bus address bus
sys_hburst[2:0]	output	System bus burst type
sys_hprot[3:0]	output	System bus protection control
sys_hsize[2:0]	output	System bus transfer size
sys_htrans[1:0]	output	System bus transfer type
sys_hwdata[SYS_DATA_WIDTH-1:0]	output	System bus write data bus
sys_hwrite	output	System bus transfer direction

Continued on next page...

Table 136: (continued)

Signal Name	Direction	Description
sys_hbusreq	output	System bus bus request

Table 137: AHB Master Transactions Used by NCEPLDM200 Bus Access

Configuration	Transaction Type
SYS_DATA_WIDTH == 32	SINGLE BYTE
	SINGLE HALFWORD
	SINGLE WORD
SYS_DATA_WIDTH == 64	SINGLE BYTE
	SINGLE HALFWORD
	SINGLE WORD
	SINGLE DOUBLEWORD
SYS_DATA_WIDTH == 128	SINGLE BYTE
	SINGLE HALFWORD
	SINGLE WORD
	SINGLE DOUBLEWORD
	SINGLE QUADWORD

Table 138: AXI Slave Signals of NCEPLDM200

Signal Name	Direction	Description
rv_awid[3:0]	input	Processor write address ID
rv_awaddr[ADDR_WIDTH-1:0]	input	Processor write address
rv_awlen[7:0]	input	Processor write burst length
rv_awsize[2:0]	input	Processor write burst size
rv_awburst[1:0]	input	Processor write burst type
rv_awlock	input	Processor write lock type
rv_awcache[3:0]	input	Processor write cache type
rv_awprot[2:0]	input	Processor write protection type
rv_awvalid	input	Processor write address valid
rv_awready	output	Processor write address ready
rv_wdata[DATA_WIDTH-1:0]	input	Processor write data
rv_wstrb[(DATA_WIDTH/8)-1:0]	input	Processor write strobes

Continued on next page...

Table 138: (continued)

Signal Name	Direction	Description
rv_wlast	input	Processor write last
rv_wvalid	input	Processor write valid
rv_wready	output	Processor write ready
rv_bid[3:0]	output	Processor write response ID
rv_bresp[1:0]	output	Processor write response
rv_bvalid	output	Processor write response valid
rv_bready	input	Processor write response ready
rv_arid[3:0]	input	Processor read address ID
rv_araddr[ADDR_WIDTH-1:0]	input	Processor read address
rv_arlen[7:0]	input	Processor read burst length
rv_arsize[2:0]	input	Processor read burst size
rv_arburst[1:0]	input	Processor read burst type
rv_arlock	input	Processor read lock type
rv_arcache[3:0]	input	Processor read cache type
rv_arprot[2:0]	input	Processor read protection type
rv_arvalid	input	Processor read address valid
rv_arready	output	Processor read address ready
rv_rid[3:0]	output	Processor read ID tag
rv_rdata[DATA_WIDTH-1:0]	output	Processor read data
rv_rresp[1:0]	output	Processor read response
rv_rlast	output	Processor read last
rv_rvalid	output	Processor read valid
rv_rready	input	Processor read ready

Table 139: RV AXI Slave Transactions Acceptable by NCEPLDM200

Configuration	AxBURST	AxLEN	AxSIZE
DATA_WIDTH == 32	INCR FIXED	0	WORD
DATA_WIDTH == 64	INCR FIXED	0	WORD DOUBLEWORD

Table 140: AXI Master Signals of NCEPLDM200

Signal Name	Direction	Description
sys_awid[3:0]	output	System bus write address ID
sys_awaddr[SYS_ADDR_WIDTH-1:0]	output	System bus write address
sys_awlen[7:0]	output	System bus write burst length
sys_awsize[2:0]	output	System bus write burst size
sys_awburst[1:0]	output	System bus write burst type
sys_awlock	output	System bus write lock type
sys_awcache[3:0]	output	System bus write cache type
sys_awprot[2:0]	output	System bus write protection type
sys_awvalid	output	System bus write address valid
sys_awready	input	System bus write address ready
sys_wdata[SYS_DATA_WIDTH-1:0]	output	System bus write data
sys_wstrb[(SYS_DATA_WIDTH/8)-1:0]	output	System bus write strobes
sys_wlast	output	System bus write last
sys_wvalid	output	System bus write valid
sys_wready	input	System bus write ready
sys_bid[3:0]	input	System bus write response ID
sys_bresp[1:0]	input	System bus write response
sys_bvalid	input	System bus write response valid
sys_bready	output	System bus write response ready
sys_arid[3:0]	output	System bus read address ID
sys_araddr[SYS_ADDR_WIDTH-1:0]	output	System bus read address
sys_arlen[7:0]	output	System bus read burst length
sys_arsize[2:0]	output	System bus read burst size
sys_arburst[1:0]	output	System bus read burst type
sys_arlock	output	System bus read lock type
sys_arcache[3:0]	output	System bus read cache type
sys_arprot[2:0]	output	System bus read protection type
sys_arvalid	output	System bus read address valid
sys_arready	input	System bus read address ready
sys_rid[3:0]	input	System bus read ID tag
sys_rdata[SYS_DATA_WIDTH-1:0]	input	System bus read data
sys_rresp[1:0]	input	System bus read response
sys_rlast	input	System bus read last

Continued on next page...

Table 140: (continued)

Signal Name	Direction	Description
sys_rvalid	input	System bus read valid
sys_rready	output	System bus read ready

Table 141: AXI Master Transactions Used by NCEPLDM200 Bus Access

Configuration	AxBURST	AxLEN	AxSIZE
SYS_DATA_WIDTH == 32	FIXED	0	BYTE HALFWORD WORD
SYS_DATA_WIDTH == 64	FIXED	0	BYTE HALFWORD WORD DOUBLEWORD
SYS_DATA_WIDTH == 128	FIXED	0	BYTE HALFWORD WORD DOUBLEWORD QUADWORD

Table 142: External Trigger Signals of NCEPLDM200

Signal Name	Direction	Description
xtrigger_halt_in[DMXTRIGGER_COUNT-1:0]*	input	External group halt request (level)
xtrigger_halt_out[DMXTRIGGER_COUNT-1:0]*	output	External group halt acknowledge (pulse)
xtrigger_resume_in[DMXTRIGGER_COUNT-1:0]*	input	External group resume request (level)
xtrigger_resume_out[DMXTRIGGER_COUNT-1:0]*	output	External group resume acknowledge (pulse)

Note

- When `DMXTRIGGER_COUNT` is 0, the width of the IO ports for external triggers is set to 1 bit.
- `xtrigger_halt_in` and `xtrigger_resume_in` should be clocked in the same clock domain as NCEPLDM200.
- The external input signals are treated as level signals while the output signals are valid for a single cycle only.

20.5.20 System Bus Access

The system bus access feature is used for read or write transactions from the debug module to the system bus. Some example sequences for bus read and bus write are provided as follows:

READ SINGLE ACCESS SIZE FROM MEMORY:

1. Set `sbcs.sbreadonaddr` = 1 and `sbcs.sbreadondata` to 0.
2. Check if `sbcs.sbbusy` is 0 before starting a new transaction.
3. Write `sbaddress0` with the target address.
   - Because `sbcs.sbreadonaddr` is 1, the write to `sbaddress0` will automatically trigger the read transaction on the system bus and update `sbdata0` accordingly.
4. Check if `sbcs.sbbusy` is 0 (i.e. transaction is done.)
5. Read `sbdata0` back to the debug host.

READ BLOCK OF MEMORY:

1. Set `sbcs.sbreadonaddr` to 1, `sbcs.sbautoincrement` to 1 and `sbcs.sbreadondata` to 1.
2. Check if `sbcs.sbbusy` is 0 before starting a new transaction.
3. Write `sbaddress0` with the target address.
   - Because `sbcs.sbreadonaddr` is 1, the write to `sbaddress0` will automatically trigger the read transaction on the system bus and update `sbdata0` accordingly.
4. Check if `sbcs.sbbusy` is 0 (i.e. transaction is done.)
5. Read `sbdata0` back to debug host.
   - Because `sbcs.sbreadondata` is 1, the read from `sbdata0` will return the previously updated data and automatically trigger the next read transaction with the increased `sbaddress0` on system bus.

6. Repeat Step-4 to Step-5 until the sequential read is finished.

WRITE SINGLE ACCESS SIZE FROM MEMORY:

1. Check if `sbcs.sbbusy` is 0 before starting a new transaction.
2. Write `sbaddress0` with the target address.
3. Write `sbdata0` with the target data.
   - The write to `sbdata0` will automatically trigger the write transaction on the system bus.
4. Check if `sbcs.sbbusy` is 0 (i.e. transaction is done.)

WRITE BLOCK OF MEMORY:

1. Set `sbcs.sbautoincrement` to 1
2. Check if `sbcs.sbbusy` is 0 before starting a new transaction.
3. Write `sbaddress0` with the target access.
4. Write `sbdata0` with the target data.
   - The write to `sbdata0` will trigger the write transaction on system bus and then automatically increase `sbaddress0` for the next write transaction.
5. Check if `sbcs.sbbusy` is 0 (i.e. transaction is done.)
6. Repeat Step-4 to Step-5 until the sequential write is finished.

20.5.21 Non-Polling Access to Debug Module

Under debug mode, a core keeps fetching and executing from NCEPLDM200 without the Non-polling feature. The Non-polling feature reduces unnecessary polling accesses from a core to NCEPLDM200 by entering a wait-state when a debug command has been serviced and waits until there are new commands available in NCEPLDM200. NCEPLDM200 informs the target core to leave wait-state via `debugint` when there is any new command requested by the external debugger.

Legacy AndesCore processors without the non-polling access feature can still work well with NCE-PLDM200. It just keeps polling continuously for the availability of the next abstract command.

20.5.22 Group Halting

The group halting feature allows halting harts in groups. By default, all the harts belong to the halt group 0, a special group that does cause the harts in this group to halt in groups.

The number of halt groups besides group 0 is specified by configuring the HALTGROUP_COUNT parameter. To verify the presence of a halt group, specify a value to group in dmcs2, then read back and compare the value to confirm the existence of the specified group.

When a hart is halted, all the harts in the same halt group will also be halted as soon as possible. If a hart is halted by group halting during resume process, it will enter debug mode again right after the resume process is finished.

Though executing quick access commands will automatically halt the selected hart, it is not considered as a halt event, so the other harts in the same group will not be halted. If a hart is halted by a quick access command before enabling group halting, it will be halted again right after the command is finished.

20.5.23 Group Resume

The group resume feature allows resuming harts in groups. By default, all the harts belong to the resume group 0, a special group that does not cause the harts in this group to halt in groups.

The number of resume groups besides group 0 is specified by configuring the HALTGROUP_COUNT parameter, so that number of halt groups and resume groups are the same. To verify the presence of a resume group, specify a value to group in dmcs2, then read back and compare the value to confirm the existence of the specified group.

Although the set of harts in a halt group and a resume group can be different, it is not recommended. It is advised that a hart should be in the same halt and resume group to avoid confusions.

When a hart is resumed, all harts in the same resume group will also be resumed. However, NCE-PLDM200 executes the resume action sequentially, one hart at a time, to simplify the hardware logic and avoid corner cases. This group resume feature only saves the JTAG iteration cycles needed by the external debugger to resume each hart one at a time. Please note that when a hart is stopped, the external debugger may corrupt its architectural states when it is in the halt mode. The external debugger still has to restore the architectural states one by one before allowing group resume to trigger.

20.6 External Triggers

External triggers are interface signals to signal halt/resume activity from another debug domain. Each configured external trigger consists of a halt in/out pair (xtrigger_halt_in/out) and a resume in/out pair (xtrigger_resume_in/out). The input signals serve as requests to trigger the group halt/resume actions and the output signals serve as acknowledge signals reflecting the group event getting triggered for the respective input signal.

All external triggers are assigned to group 0 by default, which means they do not participate in any group halt/resume actions and no cross trigger action will be performed. An external trigger needs to be programmed to join halt/resume groups to trigger group halt/resume events in the debug module. Please see dmcs2 for details.

20.7 NCEJDTM200

NCEJDTM200 is an implementation of the JTAG debug transport module (DTM), as defined by the spec: *RISC-V External Debug Support (TD003) V0.13*. It implements the IEEE 1149.1 style test access port controller (TAP). The supported instructions are summarized in Table 143.

Table 143: Supported TAP Instructions of NCEJDTM200

Encoding	Instruction	Definition
b11111	BYPASS	Section 20.7.2
b00001	IDCODE	Section 20.7.3
b10000	dtmcs	Section 20.7.4
b10001	dmi	Section 20.7.5

20.7.1 Interface Signals

Table 144: NCEJDTM200 Interface Signals

Signal Name	Direction	Description
pwr_rst_n	Input	Power on reset for NCEJDTM200
tck	Input	JTAG TCK clock
tms	Input	JTAG TMS signal
tms_out_en	Output	JTAG TMS output enable
tdi	Input	JTAG TDI signal
tdo	Output	JTAG TDO signal
test_mode	Input	test_mode should be asserted (Active-High) during the entire session of ATPG test mode.
dmi_hresetn	Output	The reset signal for NCEPLDM200
dmi_hclk	Input	The clock signal for DMI
dmi_hsel	Output	DMI selection
dmi_haddr	Output	DMI address bus

Continued on next page...

Table 144: (continued)

Signal Name	Direction	Description
dmi_htrans	Output	DMI transfer type
dmi_hsize	Output	DMI transfer size
dmi_hburst	Output	DMI burst type
dmi_hprot	Output	DMI protection control
dmi_hwrite	Output	DMI transfer direction
dmi_hwdata	Output	DMI write data bus
dmi_hrdata	Input	DMI read data bus
dmi_hresp	Input	DMI transfer response
dmi_hready	Output	DMI transfer done of NCEJDTM200
dbg_wakeup_req	Output	System wakeup request

20.7.2 BYPASS

When the TAP instruction is BYPASS, a single-bit register is connected to *tdi* and *tdo*. In Capture-DR state, the register is loaded by 0. In Shift-DR state, data is transfered from *tdi* to *tdo* through the single-bit register.

20.7.3 IDCODE

This register contains device identification code: 0x1000563D.

31 28	27 12	11 1	0
Version	PartNumber	ManufId	1

Field Name	Bits	Description	Type	Reset
ManufId	[11:1]	Identifies the designer/manufacturer of this part.	RO	0x31E
PartNumber	[27:12]	Identifies the designer's part number of this part.	RO	0x0005
Version	[31:28]	Identifies the release version of this part.	RO	0x1

20.7.4 DTM Control and Status (`dtmcs`)

31–18	17	16	15	14–12	11–10	9–4	3–0
0	dmihardreset	dmireset	0	idle	dmistat	abits	Version

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>Version</td><td>[3:0]</td><td>Version of the implemented DTM. 0x1 indicates that the current implementation conforms to RISC-V External Debug Support (TD003) V0.13.</td><td>RO</td><td>0x1</td></tr>
<tr><td>abits</td><td>[9:4]</td><td>Bit width of DMI address is 7</td><td>RO</td><td>0x7</td></tr>
<tr><td>dmistat</td><td>[11:10]</td><td>State of DMI<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No error</td></tr><tr><td>1</td><td>Reserved</td></tr><tr><td>2</td><td>An operation failed (resulted in op of 2)</td></tr><tr><td>3</td><td>An operation was attempted while a DMI access was still in progress (resulted in op of 3)</td></tr></table></td><td>RO</td><td>0x0</td></tr>
<tr><td>idle</td><td>[14:12]</td><td>This is a hint to the debugger of the minimum number of cycles a debugger should spend in RunTest/Idle after every DMI scan to avoid a busy return code (dmistat of 3).</td><td>RO</td><td>0x7</td></tr>
<tr><td>dmireset</td><td>[16]</td><td>Writing 1 to this bit clears the sticky error state and allows the DTM to retry or complete the previous transaction.</td><td>W1</td><td>0</td></tr>
<tr><td>dmihardreset</td><td>[17]</td><td>Writing 1 to this bit does a hard reset of the DTM, causing the DTM to forget about any outstanding DMI transactions. In general, this should only be used when the debugger has reasons to expect that the outstanding DMI transaction will never complete (e.g., a reset condition causes an inflight DMI transaction to be canceled).</td><td>W1</td><td>0</td></tr>
</table>

20.7.5 Debug Module Interface Access (`dmi`)

The Debug Module Interface (DMI) is accessed through this register.

40–34	33–2	1–0
address	data	op

<table>
<tr><th>Field Name</th><th>Bits</th><th>Description</th><th>Type</th><th>Reset</th></tr>
<tr><td>op</td><td>[1:0]</td><td>Write operation:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Ignore data and address. (nop)</td></tr>
<tr><td>1</td><td>Read from address. (read)</td></tr>
<tr><td>2</td><td>Write data to address. (write)</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table>
Read operation:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>The previous operation completed successfully.</td></tr>
<tr><td>1</td><td>Reserved</td></tr>
<tr><td>2</td><td>A previous operation failed.</td></tr>
<tr><td>3</td><td>An operation was attempted while a DMI request is still in progress. The data scanned into dmi in this access will be ignored.</td></tr>
</table>
</td><td>RW</td><td>2</td></tr>
<tr><td>data</td><td>[33:2]</td><td>The data to send to the DM over the DMI during Update-DR, and the data returned from the DM as a result of the previous operation.</td><td>RW</td><td>0</td></tr>
<tr><td>address</td><td>[40:34]</td><td>Address used for DMI access. In Update-DR this value is used to access the DM over the DMI.</td><td>RW</td><td>0</td></tr>
</table>

20.7.6 Debug Wake Up Request (`dbg_wakeup_req`)

This signal is typically connected to the system management unit (SMU) to allow NCEPLDM200 to be powered off in the normal system operation unless an external debugger ever connects to the chip.

Once JTAG controller leaves the Test-Logic-Reset state, this signal will be set to indicate that the external debugger has connected to the JTAG controller. All debug related logics should therefore be powered on to handle debug requests from the external debugger. Once this signal is set, it can only be cleared through power-cycling.

20.8 Programming Sequences for the External Debugger

The appendix B of the RISC-V Debug Specification (https://github.com/riscv/riscv-debug-spec) describes the typical steps to access the debugger.

All Debug operations are carried out through read/writes to the DMI memory space that are directly attached to the debug transport module (DTM, e.g., NCEJDTM200, the JTAG Debug Transport Module). The following subsections summarize the sequences to carry out typical debug operations.

20.8.1 Debug Module Interface Access

The Debug Module Interface (DMI) is a 32-bit bus. It can be accessed through NCEJDTM200 using the dmi JTAG command (IR=0x11). Typical sequence to access the DMI bus through NCEJDTM200 involves:

- Set IR=0x01 to submit the idcode command to confirm that the device identification code for the JTAG controller is 0x1000563D.
- Set IR=0x11 and set DR accordingly to submit the dmi command for accessing the DMI bus.
  - set dmi.ADDRESS to the DMI address of the desired control register.
  - set dmi.DATA to the desired data if this is a write operation.
  - set dmi.OP to read or write
- Set IR=0x10 to submit the dtmcs command to check for errors in dtmcs.DMISTAT.
  - Typically no errors should occur.
  - If DMI commands are submitted too fast without proper delay in between, dtmcs.DMISTAT will be 3.

The `dmi` commands are usually submitted by the external debugger in batches, with a predetermined delay interval to avoid overrun, with a final check of dtmcs.DMISTAT to confirm that no overrun errors occur. On overrun errors, dtmcs.DMISTAT is sticky and it can be cleared by writing one to dtmcs.DMIRESET or dtmcs.DMIHARDRESET. Reset of the tap controller also clears dtmcs.DMISTAT.

20.8.2 Activating the Debug Module

The DMACTIVE field of dmcontrol should be set to activate the debug module. This flag should remain set for the entire duration of external debugger operations.

20.8.3 Selecting the Hart to Debug

The HARTSEL field of dmcontrol should be set to select the hart to debug.

Note

It is possible to select multiple harts and halt them at the same time. Multi-hart selection is done through hawindowsel and hawindow instead of dmcontrol.HARTSEL. However, all abstract commands described in subsequent sections only apply to the hart specified by dmcontrol.HARTSEL.

20.8.4 Halting

- The HALTREQ field of dmcontrol should be set to send a debug interrupt to the selected hart.
- dmstatus should be polled to wait for the selected hart to be halted:
  - dmstatus.ALLHALTED should be checked to confirm that the selected hart is halted.
  - dmstatus.ANYUNAVAIL should be checked in case the selected hart cannot respond to the debug interrupt (e.g., power-off)
  - dmstatus.ANYNONEXISTENT can be checked if the selected hart does not exist in the system.

20.8.5 Running (Resume)

- The RESUMEREQ field of dmcontrol should be set to cause the selected hart to resume.
- dpc could be adjusted to control the resume PC.
- dcsr.PRV could be adjusted to control the privilege level to resume.

20.8.6 Single Step

- dcsr.STEP should be set before resuming the selected hart. Only one instruction will be executed before re-entering Debug Mode. The executed instruction may raise an except instead of being retired.
- dcsr.STEPIE can be set to disable interrupts for the instruction being single-stepped over.

20.8.7 Accessing Registers

The selected hart must be halted before the external debugger can access its registers. To read a register (any of general purpose registers, floating-point registers or CSRs):

- Write to the command register to initiate an *abstract command*
  - command.CMDTYPE should be set to 0 for abstract commands.
  - command.REGNO should be set to the index of the desired register (see Table 145).
  - command.WRITE should be cleared for read operations.
  - command.SIZE should be set to 2 or 3 depending on XLEN.
- Wait for the abstract command to finish by polling abstractcs
  - abstractcs.CMDERR should be 0. Note that this field is sticky and it needs to be cleared when set.
  - abstractcs.BUSY should be 0.
- Collect the read data through the Abstract Data registers.
  - The higher part of the data is in data1 if XLEN==64 of FLEN=64.
  - The lower part of the data is in data0.

Note

Both abstractcs.CMDERR and abstractcs.BUSY should be 0 before submission of abstract register-read commands for the commands to be executed successfully.

The index of registers are defined as follows:

Table 145: Abstract Registers Numbers

Numbers	Group Definition
0x0000–0x0fff	CSRs. The "PC" can be accessed here through dpc.
0x1000–0x101f	GPRs.
0x1020–0x103f	Floating point registers.

To Write a register (any of general purpose registers, floating-point registers or CSRs):

- Prepare the data to write in the Abstract Data registers.
  - Write the higher part of the data to data1 if XLEN==64 or FLEN==64
  - Write the lower part of the data to data0
- Write to command register to initiate an *abstract command*
  - command.CMDTYPE should be set to 0 for abstract commands.
  - command.REGNO should be set to the index of the desired register (see Table 145.
  - command.WRITE should be set for write operations.
  - command.TRANSFER should be set for write operations.
  - command.SIZE should be set to 2 or 3 depending on XLEN or FLEN.
- Wait for the abstract command to finish by polling abstractcs
  - abstractcs.CMDERR should be 0. Note that this field is sticky and it needs to be cleared when set.
  - abstractcs.BUSY should be 0.

Note

Both abstractcs.CMDERR and abstractcs.BUSY should be 0 before submission of abstract register-write commands for the commands to be executed successfully.

20.8.8 Accessing Memory

The debug module offers two ways of accessing memory. The first one is through the selected hart. This way of memory access will inherent the memory view of the selected hart (i.e., see/update the target line in D-Cache). The selected hart must be halted to carry out these operations. This section discuss the procedures to setup these type of memory accesses. The other way is through direct system bus access, which is described in the next section.

To read data through the selected hart (such that the latest data in D-Cache can be accessed):

- Write the desired address to *arg1* of the Abstract Data registers:
  - data1 if XLEN=32
  - data3 and data2 if XLEN=64
- Write to command register to initiate an *abstract command*
  - command.CMDTYPE should be set to 2 for abstract commands.
  - command.AAMSIZE should be set according to the size of desired memory access
- Wait for the abstract command to finish by polling abstractcs
  - abstractcs.CMDERR should be 0. Note that this field is sticky and it needs to be cleared when set.
  - abstractcs.BUSY should be 0.
- Collect the read data through the *arg0* of the Abstract Data registers.
  - The higher part of the data is in data1 if size of the access is 64-bit.
  - The lower part of the data is in data0.

Note

- Both abstractcs.CMDERR and abstractcs.BUSY should be 0 before submission of these abstract memory-read commands for the commands to be executed successfully.
- D-Cache policy for load operations in the debug mode is write-no-allocate to minimize disruptions to the cache content.

To Write data through the selected hart (such that the latest data in D-Cache can be updated):

- Write the desired address to *arg1* of the Abstract Data registers:
  - data1 if XLEN=32
  - data3 and data2 if XLEN=64
- Prepare the data to write in *arg0* of the Abstract Data registers.
  - Write the higher part of the data to data1 if the desired memory access is 64-bit.
  - Write the lower part of the data to data0.
- Write to command register to initiate an *abstract command*
  - command.CMDTYPE should be set to 0 for abstract commands.
  - command.AAMSIZE should be set according to the size of desired memory access
  - command.WRITE should be set for write operations.
- Wait for the abstract command to finish by polling abstractcs
  - abstractcs.CMDERR should be 0. Note that this field is sticky and it needs to be cleared when set.
  - abstractcs.BUSY to be 0.

Note

- Both abstractcs.CMDERR and abstractcs.BUSY should be 0 before submission of these abstract memory-write commands for the commands to be executed successfully.
- D-Cache policy for store operations in the debug mode is write-through-no-allocate to minimize disruptions to the cache content.

20.8.9 Direct System Bus Memory Access

The system bus memory access operations are performed by the debug module initiating system bus accesses directly. It can be used to bypass caches of the target hart. The system bus memory access also provides 128-bit data access if it is available to the debug module.

To read data directly through system bus memory access:

- Setup sbcs:
  - Set sbcs.SBACCESS to specify the size of the desired bus access.
  - Set sbcs.SBREADONADDR to specify that a bus read should be triggered on every write to sbaddress0.
- Write the target address to sbaddress
  - Write the higher part of the address to sbaddress1 first if address width of the system bus (sbcs.SBASIZE) is larger than 32.
  - Write the lower part of the address to sbaddress0 the last, as updating it will start the bus read access because sbcs.SBREADONADDR is set.
- Wait for the bus access to finish by polling sbcs
  - sbcs.SBERROR should be 0. Note that this bit is sticky and it needs to be cleared when set.
  - sbcs.SBBUSYERROR should be 0. Note that this bit is sticky and it needs to be cleared when set.
  - sbcs.SBBUSY should be 0.
- Collect data from sbdata:
  - sbdata0 if sbcs.SBACCESS is less than or equal to 32-bits
  - sbdata1 and sbdata0 if sbcs.SBACCESS is 64-bit.
  - sbdata3 .. sbdata0 if sbcs.SBACCESS is 128-bit.

Note

- All of sbcs.SBERROR, sbcs.SBBUSYERROR and sbcs.SBBUSY should be 0 before submission of these system bus memory reads for the operations to be executed successfully.

To write data directly through system bus memory access:

- Setup sbcs:
  - Set sbcs.SBACCESS to specify the size of the desired bus access.
  - Clear sbcs.SBREADONADDR for memory write operations.
- Write the target address to sbaddress
  - Write the higher part of the address to `sbaddress1` if address width of the system bus (`SBASIZE`) is larger than 32.
  - write the lower part of the address to sbaddress0.
- Write data to sbdata and starts the bus write access.
  - Write the higher word of the data to sbdata1 if sbcs.SBACCESS is 128-bit.
  - Write bit 63-32 of the data to sbdata1 if sbcs.SBACCESS is 64-bit.
  - Write the lower part of the data to sbdata0; Writing to this register starts the bus write.
- Wait for the bus access to finish by polling sbcs
  - sbcs.SBERROR should be 0. Note that this bit is sticky and it needs to be cleared when set.
  - sbcs.SBBUSYERROR should be 0. Note that this bit is sticky and it needs to be cleared when set.
  - sbcs.SBBUSY should be 0.

Note

- All of sbcs.SBERROR, sbcs.SBBUSYERROR and sbcs.SBBUSY should be 0 before submission of these system bus memory writes for the operations to be executed successfully.

21 Andes Custom Extension (ACE)

This section describes required integration efforts needed for integrating ACE with the base NX25(F) design.

21.1 Generated Files for ACE

If ACE memories (ACM) are used, COPILOT will modify top-level files to generate pre-integrated ACM signals, including `nx25_core.v`, `vc_core.v`, `acm_subsystem.v`, and `ae350_cpu_subsystem.v`. The top-level files are under **$NDS_HOME**/andes_ip/vc_core/top/hdl/. The rest of generated ACE design files must be copied to the proper places to integrate with the base NX25(F) design. The `install_ace_files` script is provided to perform the proper copying. Note that the old files will be removed first by `install_ace_files` before copying. The `install_ace_files` script can be found as

$NDS_HOME/tools/bin/install_ace_files

The following table shows the destination directories where the ACE generated files are relocated to:

Generated ACE Files	Target Directory	Description
./rtl/hdl/*.v	**$NDS_HOME**/andes_ip/vc_core/ace/hdl	ACE design files
./rtl/memory/fpga/*.v	**$NDS_HOME**/andes_ip/vc_core/ace/memory/fpga	ACM memory macro files for synthesis (only when an ACM is used)
./rtl/memory/model/*.v	**$NDS_HOME**/andes_ip/vc_core/ace/memory/model	ACM memory macro files for synthesis (only when an ACM is used)
./rtl/memory/syn/*.v	**$NDS_HOME**/andes_ip/vc_core/ace/memory/syn	ACM memory macro files for synthesis (only when an ACM is used)
./rtl/syn/ace_flist.tcl	**$NDS_HOME**/andes_ip/vc_core/syn/script	ACE design file list (for DC)
./rtl/syn/ace_flist.tcl	**$NDS_HOME**/andes_ip/vc_core/syn/script_rc	ACE design file list (for RC)

21.2 Models for ACE

The model usage for ACE is as the assumption described in Section 22. The design hierarchy is as illustrated in Figure 2 and Figure 4; SRAM-ACM is instantiated in the CPU subsystem level. As for AHB-ACMs, COPILOT only generates and propagates AHB interface signals up to the CPU subsystem level module. Integration of these interfaces of AHB-ACMs should be done according to the system requirements.

When SRAM-ACMs are used, the sample module with the behavioral SRAM model instantiation can be found as `./rtl/memory/model/ace_sramN.v`, where N is a number starting from 0. Template files for synthesis and FPGA can be found as `./rtl/memory/syn/ace_sramN.v` and `./rtl/memory/fpga/ace_sramN.v`, respectively. Proper memory macro must be generated and instantiated based on the ACM configuration. The corresponding ACM information will be shown in the template files, including the name for an SRAM-ACM and widths of the address bus and data bus. The following listing is an example for a 32x128-sized SRAM-ACM named "coeff": 32x128:

```
module ace_sram0( // coeff: 32x128
    core_clk,
    ace_sram_we,
    ace_sram_cs,
    ace_sram_addr,
    ace_sram_din,
    ace_sram_dout
);

localparam      ADDR_WIDTH = 5;
localparam      DATA_WIDTH = 128;

input                   core_clk;
input                   ace_sram_we;
input                   ace_sram_cs;
input  [ADDR_WIDTH-1:0] ace_sram_addr;
input  [DATA_WIDTH-1:0] ace_sram_din;
output [DATA_WIDTH-1:0] ace_sram_dout;

//
// Proper memory macro must be generated and instantiated accordingly.
//
// ------------------------------
// | ACM name: coeff
// | - address_bits: 5 (32-entry)
```

```
// | - width: 128
// ------------------------------


endmodule
```


21.3 Simulation with ACE

Please follow the instructions and steps in the *RTL Simulation on CPU Product Package* chapter of the *Andes Custom Extension User Manual* to install the ACE sample test cases to the directory of the NX25(F) distribution and to run the RTL simulation.

21.4 Synthesis with ACE

To add timing constraints for ACE Custom Logic, please modify the constraint file:

(Cadence Genus) **$NDS_HOME**/andes_ip/vc_core/syn/script_rc/timing_con.tcl

(Synopsys DC) **$NDS_HOME**/andes_ip/vc_core/syn/script/timing_con.tcl

It is assumed that all ACM interfaces are connected within the ae350_cpu_subsystem module. However, if new I/O interface signals are created to connect these interfaces outside the ae350_cpu_subsystem module, please modify the I/O constraint file:

(Cadence Genus) **$NDS_HOME**/andes_ip/vc_core/syn/script_rc/io_delay.tcl

(Synopsys DC) **$NDS_HOME**/andes_ip/vc_core/syn/script/io_delay.tcl

22 Models

NX25(F) requires several memory cells. Behavioral SRAM models are instantiated for these memory cells in the release package. These memory cells should be replaced with the cells in your library based on the actual configuration. Figure 2 and Figure 4 illustrate the SRAM models connected to the NX25(F) processor design.

Reference memory instantiations with behavioral SRAM models can be found in the **$NDS_HOME**`/andes_ip/vc_core/memory/model` directory. They are only good for simulations, and they should be replaced by files that instantiate the SRAM macros from the targeted technology library. It is suggested that a dedicated new directory **$NDS_HOME**`/andes_ip/vc_core/memory/syn` is created to hold the synthesizable copy of these memory instantiations.

22.1 Important Assumptions on SRAMs

All SRAMs used by NX25(F) are single cycle latency: all control bits (chip-selects, addresses and write-enable) and write-data appear on the interface in the first cycle and (read) data returns on the next cycle. Please note that in case a single SRAM needs to be composed of multiple smaller SRAMs, the select signals for the read data selection mux need to be coming from flopped version of the address bus in order to align to the data cycle.

The data output ports of SRAMs are expected to hold the values from the most recent accesses. For SRAMs that do not hold the output value, wrappers must be added to remember the output values.

Combining the previous two requirements, the flopped version of the address bus for muxing data among outputs of smaller SRAMS should only be updated when the top chip select signal is asserted. Otherwise, a free-running flop for the delayed version of the address bus may toggle and change the value of output data when chip select is not asserted. Figure 26 shows how the output data mux should be implemented for a SRAM module is composed of two smaller SRAM cells.

```
assign bank0_cs = cs & addr[N];
assign bank1_cs = cs & ~addr[N];
assign rdata = data_select ? bank1_rdata : bank0_rdata
;

always @(posedge clk) begin
        if (cs) begin
                data_select <= addr[N];
        end
end
sram bank0(
        .cs (bank0_cs),
        .rdata (bank0_rdata),
        ...
);
sram bank1(
        .cs (bank1_cs),
        .rdata (bank1_rdata),
        ...
```

Figure 26: Output Muxing for Composing a Larger SRAM with Smaller Ones

22.2 Branch Target Buffer (BTB) Organization

BTB is implemented as a two-way associative array to store the branch prediction data. BTB reference memory instantiations with behavioral SRAM models can be found in the **$NDS_HOME**/andes_ip/vc_core/memory/model/vc_btb_ram.v directory.

The SRAMs for BTB (vc_btb_ram) are instantiated twice in the CPU subsystem.

The I/O ports for btb_ram.v and SRAM dimension are summarized in Section 3.14. The signals with prefix "btb0_" and "btb1_" are for BTB way 0 and way 1, respectively.

22.3 Instruction Local Memory Organization

ILM in NX25(F) consists of vanilla SRAMs. Sample `vc_ilm_ram.v` with the behavioral SRAM model instantiation can be found as **$NDS_HOME**/andes_ip/vc_core/memory/model/vc_ilm_ram.v. Please copy it to the `memory/syn` directory before modification.

The SRAM for ILM (vc_ilm_ram) is instantiated in the CPU subsystem. The I/O ports for vc_ilm_ram and its SRAM dimension are summarized in Section 3.9. The signals with prefix "ilm_" are for ILM SRAM.

22.4 Data Local Memory Organization

DLM in NX25(F) consists of vanilla SRAMs. Sample `vc_dlm_ram.v` with the behavioral SRAM model instantiation can be found as **$NDS_HOME**`/andes_ip/vc_core/memory/model/vc_dlm_ram.v`. Please copy it to the `memory/syn` directory before modification.

The SRAM for DLM (vc_dlm_ram) is instantiated in the CPU subsystem. The I/O ports for vc_dlm_ram and its SRAM dimension are summarized in Section 3.10. The signals with prefix "dlm_" are for DLM SRAM.

22.5 Instruction Cache Organization

I-Cache in NX25(F) consists of tag and data RAMs. Sample `vc_icache_ram.v` with the behavioral SRAM model instantiation can be found as **$NDS_HOME**`/andes_ip/vc_core/memory/model/vc_icache_ram.v`. Please copy it to the `memory/syn` directory before modification.

The SRAM for I-Cache (vc_icache_ram) is instantiated in the CPU subsystem. The I/O ports for vc_icache_ram and its SRAM dimension are summarized in Section 3.11. The signals with prefix "icache_" are for I-Cache SRAMs.

22.6 Data Cache Organization

D-Cache in NX25(F) consists of tag and data RAMs. Sample `vc_dcache_ram.v` with the behavioral SRAM model instantiation can be found as **$NDS_HOME**`/andes_ip/vc_core/memory/model/vc_dcache_ram.v`. Please copy it to the `memory/syn` directory before modification.

The SRAM for D-Cache (vc_dcache_ram) is instantiated in the CPU subsystem. The I/O ports for vc_dcache_ram and its SRAM dimension are summarized in Section 3.12. The signals with prefix "dcache_" are for D-Cache SRAM.

23 Simulation

23.1 Prerequisites

Please check the following items regarding your simulation environment before performing the verification:

- The NX25(F) softcore RTL requires SystemVerilog compatible simulators.
- To run simulations, the simulation environment requires standard Unix tools like `make`, `sed`, `grep` and `perl`.
- The softcore GUI configuration tool requires the Tcl/Tk interpreter `wish` to be installed in the system.
- This document assumes that environment variable **$NDS_HOME** points to the top directory of the NX25(F) distribution.
- This document assumes that environment variable **$NDS_TOOLCHAIN** points to the bin directory containing Andes toolchains.
  - Note that the Andes toolchains are not part of this distribution. Please find them in the AndeSight software development tools. This distribution ships precompiled test patterns to eliminate the dependency.

23.2 AE350 Testbench

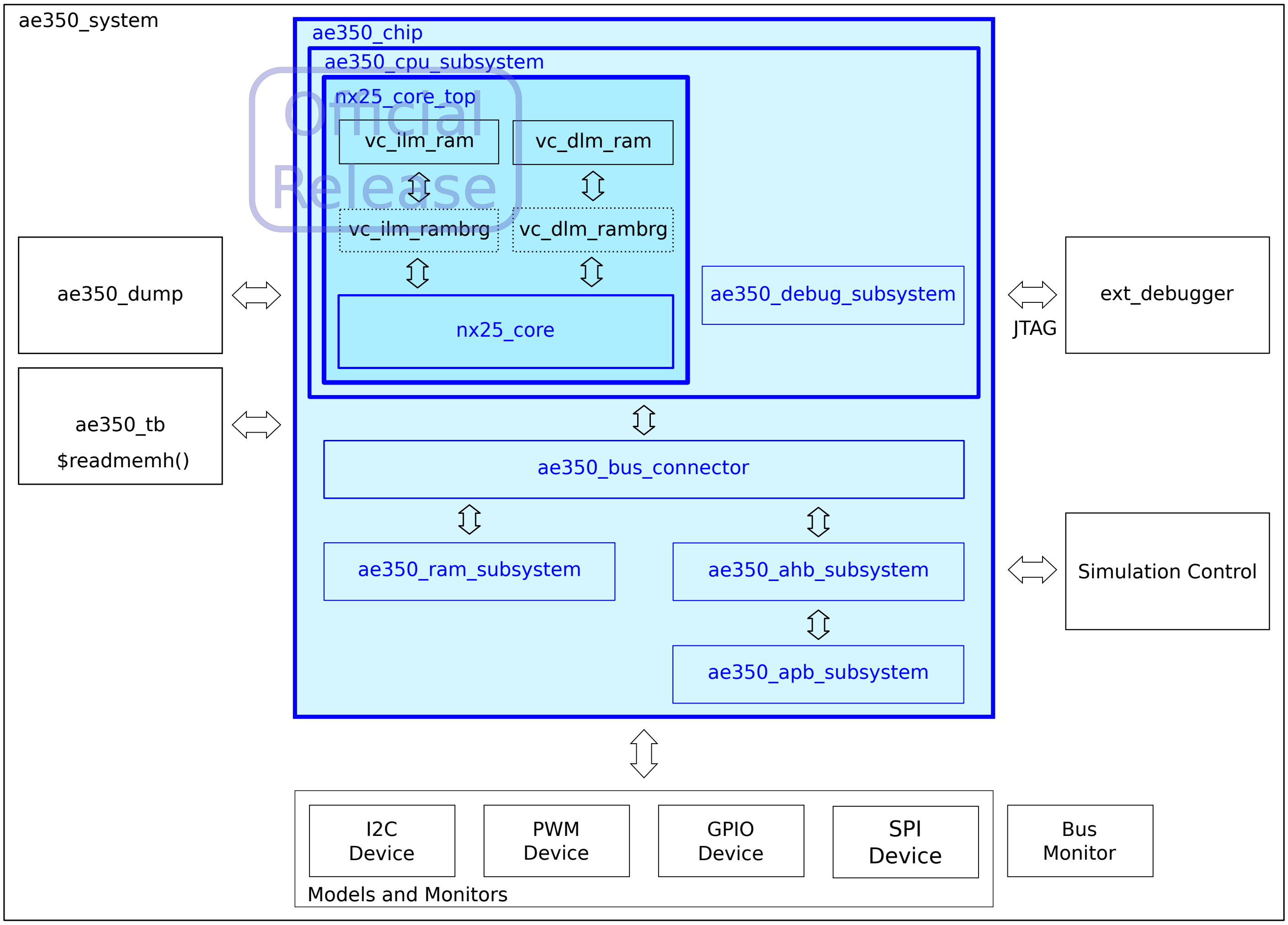


Figure 27: NX25(F) Simulation Environment

Figure 27 shows the block diagram of the NX25(F) simulation environment on the AE350 platform. The nx25_core module in the block diagram is the NX25(F) core. The ae350_tb module provides the clock sources and the reset sequences. It also loads the ROM image (`NDSROM.dat`) of the test case to ILM, ae350_ram_subsystem and the SPI device. The simulation control module (ahb_sim_control) is connected to ae350_bus_connector and is responsible mainly for the termination of simulations. The ae350_dump module controls the waveform dumping. The ext_debugger module simulates external debugger activities and interfaces with the RISC-V debug subsystem within the ae350_chip module.

23.3 Sample Test Cases

A set of sample test cases are shipped with NX25(F) for reference. The test cases are located under directory

$NDS_HOME/andes_vip/patterns/samples

23.3.1 Quick Start

A simple test bench is included in the NX25(F) distribution. To start a simulation with the generated image file,

1. Set **$NDS_HOME** to the top directory of the package:

```
bash: export NDS_HOME=<top directory of this package>
csh:  setenv NDS_HOME <top directory of this package>
```

2. Change directory to the sample pattern directory:

```
cd $NDS_HOME/andes_vip/patterns/samples
```

3. Edit Make.vars to set **$(VERILOG)** to your favorite SystemVerilog simulator.

4. Select a test case and run it:

```
cd test_power_ls
make
```

Output of the simulation should look like Figure 28. Upon a successful simulation, the “SIMULATION PASSED” string shall be observed at the end of the output file. Otherwise, simulations will either hang forever (most likely due to X-propagation) or “SIMULATION FAILED” string will appear if errors are detected.

The output message is intentionally terse to speed up simulation time. It could be decoded into assembly outputs with `ipipe_decode.pl`. The `ipipe_decode.pl` command can be found as

$NDS_HOME/tools/bin/ipipe_decode.pl

Figure 29 shows a sample decoded output.

```
linux$ make
xrun -l verilog.log -exit +licq +nowarn+CUVWSP +nowarn+LIBNOU +nowarn+SPDUSD   -f flis
t
...
68644.53 ns:ipipe:reset 80000000
...
94907.03 ns:ipipe:@00000080=0080006f
95007.03 ns:ipipe:@00000088=00200197 gp=0000000000200088
95032.03 ns:ipipe:@0000008c=77818193 gp=0000000000200800
95057.03 ns:ipipe:@00000090=00201297 t0=0000000000201090
95082.03 ns:ipipe:@00000094=f7028293 t0=0000000000201000
...
487357.00 ns:ipipe:sim_ctrl finish=0
487357.03 ns:ipipe:0:---- SIMULATION PASSED ----
...
```


Figure 28: Sample Test Case Simulation Output

```
linux$ export PATH=$NDS_HOME/tools/bin:$PATH
linux$ ipipe_decode.pl < verilog.log
...
68644.53 ns:ipipe:reset 80000000
...
94907.03 ns:ipipe:@00000080=0080006f jal zero, +0x00008 (0x00000088)
95007.03 ns:ipipe:@00000088=00200197 auipc gp, 0x00200000 (0x00200088) gp=000000000020008
8
95032.03 ns:ipipe:@0000008c=77818193 addi gp, gp, 0x778          gp=0000000000200800
95057.03 ns:ipipe:@00000090=00201297 auipc t0, 0x00201000 (0x00201090) t0=000000000020109
0
95082.03 ns:ipipe:@00000094=f7028293 addi t0, t0, -0x090         t0=0000000000201000
...
487182.00 ns:ipipe:sim_ctrl finish=0
487182.03 ns:ipipe:0:---- SIMULATION PASSED ----
```

Figure 29: `ipipe_decode.pl` Output

23.3.2 SystemVerilog Simulator Selection

The test cases are launched through Makefiles. Before starting the simulation, please edit

> **$NDS_HOME**/andes_vip/patterns/samples/Make.vars

such that the make variable **$(VERILOG)** points to a valid SystemVerilog simulator.

23.3.3 Test Case Organization

Test cases are organized as a hierarchy of directory tree through Makefiles. The default make target compiles and runs all test cases. Typing `make` at the topmost level will run all test cases under the directory. Individual test cases can also be run by starting make at the specific test case subdirectory. Examples as below:


```
# run all test cases under the "samples" subdirectory
cd $NDS_HOME/andes_vip/patterns/samples; make

# or, run test_power_ls only
cd $NDS_HOME/andes_vip/patterns/samples/test_power_ls; make
```

23.3.4 Extra Options for SystemVerilog Simulators

To pass extra options to the simulator, the **$(VPLUSDEFINES)** variable could be modified through the `make` command line or through modifying

$NDS_HOME/andes_vip/patterns/samples/Make.vars

For example, the following command could be used to enable dumping waveforms:

```
make VPLUSDEFINES="+define+DUMP+TRN +access+rc"
```

23.3.5 Simulation File List

The simulation file list is defined at:

$NDS_HOME/flists/flist.in

The `flist.in` file must be processed to expand the **$NDS_HOME** variable to the actual path value before SystemVerilog simulators could accept it as a valid command line switch file. This is handled by the following rule in `Makefile`:

```
@rm -f flist
@sed -e "s,\$$NDS_HOME,$$NDS_HOME," < $$NDS_HOME/flists/flist.in \
 | grep -v "#" | sed -e "s,//*,/,g" > flist
```

23.3.6 NDSROM.dat Image File

The ROM image file NDSROM.dat serves as the test pattern for each test case. Its format is defined by SystemVerilog $readmemh() system task. The default make target does not attempt to regenerate the ROM file. Instead, the make rom target should be used to regenerate the NDSROM.dat file.

Note

Some test patterns, such as ACE, Dhrystone and Coremark, have their own make targets for compiling and creating ROM images. Please see Section 23.3.8 for details.

For the make rom target to work, the toolchain programs (riscv64-elf-ar and riscv64-elf-gcc) should exist under the directory specified by **$NDS_TOOLCHAIN**, e.g.,

```
setenv NDS_TOOLCHAIN <directory of toolchain>/riscv64-elf-mculib/bin
```

Note that the toolchain programs (risv64-elf-ar and risv64-elf-gcc) are not included in the NX25(F) release package. Please find them in the AndeSight software development tools.

The toolchains convert the assembly/C programs to a.out files. To make the executable files loadable, a.out must be further converted into flat binary data, and then converted to the ASCII format readable by the $readmemh() SystemVerilog system task. riscv-elf-aout2mem demonstrates how that could be done for simple a.out formats with simple .text and .data sections. riscv-elf-aout2mem comes with the NX25(F) distribution and it could be found as **$NDS_HOME**/tools/bin/riscv-elf-aout2mem. It is a straightforward sample Perl script. If advanced linker sections are used, the riscv-elf-aout2mem program might need to be modified to support extra sections.

Each element of the array in NDSROM.dat is in the *big-endian* format regardless of the system endian. That is, byte 0 (0x00), 1 (0x11), 2 (0x22), and 3 (0x33) of the binary image will be represented as 0x00112233 at index 0 of the array.

23.3.7 Clean Up of Simulation Results

The target make clean can be used to clean up the simulation results.

23.3.8 Description of Test Cases

The test cases that come with this distribution could be found under the **$NDS_HOME**/andes_vip/patterns/samples directory. The test cases are described in this section.

Note

Some of the reference test cases may be designed for certain configurations only and may not work for all configurations. For example, the local memory related test cases are designed for local memory sizes larger than 4KiB and obviously they require the corresponding local memory support.
Please contact Andes Technology for further supports on specific test case issues.

test_dhrystone_v5

This pattern is the precompiled version of the Dhrystone benchmark. To compile the test pattern, please get the C source code for the Dhrystone benchmark from http://www.netlib.org/-benchmark/dhry-c, place it in the pattern directory, and type the command below to generate `NDSROM.dat` for later simulation:

```
make dhry
```

To show Dhrystone numbers, type:

```
make; make getdmips
```

test_coremark_v5

This pattern is the precompiled version of the CoreMark benchmark. The Makefile is setup to automatically get coremark from its official github source: https://github.com/eembc/coremark. Please make sure your https_proxy setting is correctly setup for git and type the command below to generate `NDSROM.dat` for later simulation:

```
make coremark
```

To show CoreMark numbers, type:

```
make; make getcoremark
```

test_whetstone_v5

This pattern is only available when FPU extension supported. It is the precompiled version of the Whetstone benchmark. To compile the test pattern, please get and port Whetstone source code from http://www.roylongbottom.org.uk/whets.c, place it in the pattern directory, and type the command below to generate `NDSROM.dat` for later simulation:

```
make whet
```

To change the floating-point precision (single or double), the **$(FPU_TEST_TYPE)** variable could be modified through the `make` command line or through modifying

```
$NDS_HOME/andes_vip/patterns/samples/test_whetstone_v5/Makefile
```

For example, the following command could be used to change to double precision:

```
make whet FPU_TEST_TYPE=dp
```

To show WIPS numbers, type:

```
make; make getwips
```


test_mem_macro
This pattern tests integrity and connectivity of instantiated memory macro.

test_meminfo
This pattern extracts information for used memory blocks in the design.

```
make getmeminfo
```

test_icache_sram
This pattern tests the connectivity of SRAM memories. This pattern touches every data and address bit of I-Cache memories.

test_dcache_sram
This pattern tests the connectivity of SRAM memories. This pattern touches every data and address bit of D-Cache memories.

test_btb_sram, test_dlm_sram, and test_ilm_sram
These patterns test the connectivity of SRAM memories. These patterns touch every data and address bit of BTB, DLM and ILM memories.

test_power_ls and test_power_mul
These patterns attempt to exercise the maximum power state of the processor.

test_caches
This pattern turns on both I-Cache and D-Cache. The test pattern causes various corners of the caches to be accessed.

test_pmp
This pattern turns on physical memory protection for both instruction fetch and data accesses.

test_debugger
This pattern demonstrates external debugger accesses.

test_wfi_resume
This pattern tests entering and leaving the WFI mode.

test_atcgpio100
This pattern tests the interrupt generated by GPIO.

test_atcpit100
This pattern tests the timer in 8/16/32-bit modes.

test_atcrtc100
This pattern tests the RTC interrupts.

test_atcwdt200
This pattern tests accesses to the registers of the watchdog timer.

test_atcapbbrg100
This pattern tests accesses to the registers of the APB bridge.

test_atcbmc300
This pattern tests accesses to the registers of the bus matrix.

test_atcuart100
This pattern tests UART read/write transactions.

test_atcspi200
This pattern tests SPI read/write transactions through register programming.

test_atcspi200_slave
This pattern tests SPI read/write transactions in the slave mode.

test_atciic100
This pattern tests I2C read/write transactions with interrupts.

test_atcdmac300
This pattern tests DMA accesses.

test_rvb
This pattern tests RISC-V bit-manipulation instructions.

test_light_sleep
This pattern tests light-sleep (clock-gated) control flow. It verifies that the core clock is gated when the core enters WFI mode. The core resumes after its clock is recovered from interrupt events.

test_deep_sleep
This pattern tests deep-sleep (power-gated) control flow. It verifies that the core is powered down while entering WFI mode. The core wakes up after its power is restored from interrupt events. For CPF or UPF low-power simulation, please see the simulator condition and command below:

- Cadence Incisive Enterprise Simulator (ncverilog/xrun) supports both CPF and UPF.
- Synopsys VCS (vcs) supports only UPF simulation.

```
# run cpf on xrun
make
# run upf on xrun
make PWR_SIM=upf
# run upf on vcs
make
```


23.3.9 Simulation Control

These patterns run in a self check manner. Upon detecting any unexpected error, the simulation will be early terminated by the program writing a specific value to ahb_sim_control to abort the simulation. Or, if everything goes fine, the program in the end writes another specific value to ahb_sim_control to gracefully terminate the simulation.

On the hardware side, ahb_sim_control achieves this by snooping AHB traffics of the internal slave (slave 0) of AHB Decoder. This internal slave is allocated a 1MiB space (see Table 102) but actually it only uses less than two hundred bytes. The `BASE` parameter of ahb_sim_control is set by default to 0x80000 so this means it will only check offset addresses behind 512KiB in this 1MiB space. This guarantees the existence of ahb_sim_control will not interfere the normal operation of this internal slave.

If the base memory address of this 1MiB space is changed, the monitored space of ahb_sim_control will also be effectively changed since the internal signals after address decoding inside AHB Decoder are directly used to do the snooping. This 1MiB space must reside in the device region and this guarantees the memory space of ahb_sim_control is also inside the device region.

When ahb_sim_control sees an AHB write transaction for the `BASE` offset with some recognized values of write data, it prints related pass/fail information and calls Verilog system task `$finish` to terminate the simulation. The control register information of ahb_sim_control is listed in Table 146.

Table 146: Simulation Control Registers

Address	I/O Type	Description
(Base address of AHB Decoder) + ahb_sim_control.BASE	Write only	Write 0x01234568 to finish simulation. Write 0x01234569 to abort simulation. Write 0x01234571 to skip simulation.

On the software side, the program calls `exit()` with the appropriate argument. `exit()` is defined inside **$NDS_HOME**/andes_vip/patterns/samples/src/ae350_isr.c. The address of ahb_sim_control is decided by macro `DEV_SIM_CONTROL` which is equal to macro `SIM_CONTROL_BASE` in value. These macros are defined inside **$NDS_HOME**/andes_vip/patterns/samples/include/ae350.h. When the memory map is changed, both hardware and software settings must still match each other.

23.4 RISC-V Verification Suite

The RISC-V verification suite is a set of unit tests provided by the RISC-V foundation that could be obtained from https://github.com/riscv/riscv-tests. A copy of the test suite is packaged and integrated in this release under the following directory to enable simulations with the NX25(F) design:

$NDS_HOME/andes_vip/patterns/riscv-tests

Some tests of the RISC-V verification suite currently fail under RISC-V configurations that they do not expect, so a set of enhancement patches is provided to fix them. Please note that the patches may have conflicts that need to be resolved when applied to the newer RISC-V verification suite.

23.4.1 Quick Start

A simple test bench is included in the NX25(F) distribution. To start a simulation with the generated image file,

1. Set **$NDS_HOME** to the top directory of the package:

```
bash: export NDS_HOME=<top directory of this package>
csh:  setenv NDS_HOME <top directory of this package>
```

2. Change directory to the directory for the RISC-V verification suite:

```
cd $NDS_HOME/andes_vip/patterns/riscv-tests
```

3. Edit Make.vars to set **$(VERILOG)** to your favorite SystemVerilog simulator.

4. Select a test case and run it:

```
cd rundir/test_rv64ui_add
make
```

Output of the simulation should look like Figure 30. Upon a successful simulation, the "SIMULATION PASSED" string shall be observed at the end of the output file. Otherwise, simulations will either hang forever (most likely due to X-propagation) or "SIMULATION FAILED" string will appear if errors are detected.

The output message is intentionally terse to speed up simulation time. It could be decoded into assembly outputs with `ipipe_decode.pl`. See Section 23.3.1 for how the script works.


```
linux$ make
xrun +licq +nowarn+CUVWSP  -l verilog.log -f flist +notiminecheck
s
...
68644.53 ns:ipipe:reset 80000000
...
95007.03 ns:ipipe:@00000044=f1402573 a0=0000000000000000
95082.03 ns:ipipe:@00000048=00051063
95107.03 ns:ipipe:@0000004c=30102573 a0=8000000000901105
95182.03 ns:ipipe:@00000050=02054063
95282.03 ns:ipipe:@00000070=00000193 gp=0000000000000000
95307.03 ns:ipipe:@00000074=00000297 t0=0000000000000074
95332.03 ns:ipipe:@00000078=f9028293 t0=0000000000000004
...
109682.00 ns:ipipe:sim_ctrl finish=0
109682.03 ns:ipipe:0:---- SIMULATION PASSED ----
...
```

Figure 30: Simulation Output for Test Case `test_rv64ui_add`

23.4.2 Updating to the Latest Test Suite

The latest RISC-V verification suite can be found at https://github.com/riscv/riscv-tests. Please copy the newer "isa" directory to replace

$NDS_HOME/andes_vip/patterns/riscv-tests/isa

After the test suite is updated, please execute `setup.sh` and generate `NDSROM.dat` again.

23.4.3 Creating Makefile and Test Case Directory

Execute `setup.sh` and it will create `Makefile` and test case directories depending on `Makefile.in` and configuration files. Please note that `setup.sh` should be executed each time the processor configuration changes or the latest test suite updates.

$NDS_HOME/andes_vip/patterns/riscv-tests/setup.sh

23.4.4 NDSROM.dat Image File

Please see Section 23.3.6 for the description of how to compile and generate the `NDSROM.dat` image file for simulation.

In addition to the descriptions in Section 23.3.6, the patch.sh script will be run to apply enhancement patches to the RISC-V verification suites when building image files by using `make rom` target. Please note that the patches may have conflicts with the newer RISC-V verification suite downloaded from Internet.

23.4.5 SystemVerilog Simulator Selection

Before starting the simulation, please edit

$NDS_HOME/andes_vip/patterns/riscv-tests/Make.vars

such that the make variable **$(VERILOG)** points to a valid SystemVerilog simulator.

Note

The Makefiles for the RISC-V verification suite do not share the same settings used by the sample test patterns described earlier in Section 23.3. So settings of all variables should be assigned separately.

23.4.6 Test Case Organization

Test cases are organized as a hierarchy of directory tree through Makefiles. The default make target compiles and runs the test cases. Typing `make` at the topmost level will run all test cases under the directory. Individual test cases can also be run by starting make at the specific test case subdirectory. Examples as below:

```
# run all test cases under the "riscv-tests/rundir" directory
cd $NDS_HOME/andes_vip/patterns/riscv-tests; make

# or, run test_rv64ui_add only
cd $NDS_HOME/andes_vip/patterns/riscv-tests/rundir/test_rv64ui_add; make
```

23.4.7 Extra Options for SystemVerilog Simulators

To pass extra options to the simulator, the **$(VPLUSDEFINES)** variable could be modified through the `make` command line or through modifying

$NDS_HOME/andes_vip/patterns/riscv-tests/Make.vars

For example, the following command could be used to enable dumping waveforms:

```
make VPLUSDEFINES="+define+DUMP+TRN +access+rc"
```

24 Synthesis of NX25(F)

There are sets of synthesis scripts to support the following tools:

- Synopsys DC (Design Compiler)
- Cadence Genus

The NX25(F) core synthesis working directory is **$NDS_HOME**/andes_ip/vc_core/syn.

24.1 Synopsys DC Synthesis

24.1.1 Introduction

The master run script that drives the entire synthesis flow is at

$NDS_HOME/andes_ip/vc_core/syn/run_syn

And it is expected to be invoked under the **$NDS_HOME**/andes_ip/vc_core/syn directory. This script will set up working directories and invoke the synthesizer.

The DC scripts and constraint files are under directory **$NDS_HOME**/andes_ip/vc_core/syn/script. The top-level synthesis script is vc_core.tcl. It calls the rest of the synthesis scripts.

The tunable TCL variables that the synthesis scripts use are collected in the following files:

$NDS_HOME/andes_ip/vc_core/syn/core_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/script/syn_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/syn_setup_dc.tcl

See Section 24.1.2 for more details.

After the synthesis completes, the results will be saved in the directories shown in the following table. These directories will be created if they do not exist.

Table 147: Synthesis Result Directories

Directory	Description
$NDS_HOME/andes_ip/vc_core/syn	Synthesis working directory
$NDS_HOME/andes_ip/vc_core/syn/ddc	Directory for DDC files
$NDS_HOME/andes_ip/vc_core/syn/log	Directory for log files
$NDS_HOME/andes_ip/vc_core/syn/netlist	Directory for netlist files
$NDS_HOME/andes_ip/vc_core/syn/rpt	Directory for synthesis reports

Continued on next page...

Table 147: (continued)

Directory	Description
$NDS_HOME/andes_ip/vc_core/syn/work	Directory for temp files generated during synthesis

24.1.2 Synthesis Environment Setup

Table 148 and Table 149 show all TCL variables that can be adjusted, and they are discussed in the following subsections.

24.1.2.1 Technology Library and Memory Macros

Technology library and memory macros are specified in

$NDS_HOME/andes_ip/vc_core/syn/syn_setup_dc.tcl

In addition, this TCL file also contains settings of technology-related TCL variables, which include operating_cond, loading_cell, driving_cell, max_trans (max transitions), dont_use_cells, and wire_load_group (wire load model selection).

24.1.2.2 Synthesis Configuration

Four configuration scripts and a power pattern needed for performing synthesis are located separately at:

$NDS_HOME/andes_ip/vc_core/syn/core_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/pf_info.tcl
$NDS_HOME/andes_ip/vc_core/syn/verilog.saif
$NDS_HOME/andes_ip/vc_core/syn/script/syn_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/script/io_delay.tcl

The `core_env.tcl` configuration script sets TCL variables related to root design, target frequencies, and I/O timing. The `pf_info.tcl` script is for power report setting. The `verilog.saif` is the power pattern. `pf_info.tcl` and `verilog.saif` are generated automatically. The `syn_env.tcl` script parses `config.inc` for processor configurations. The `io_delay.tcl` script applies the actual I/O constraints.

Note that the `core_env.tcl` script is located one level higher than other scripts to be shared by all synthesis tools.

Setting the Target Frequencies

The target frequencies are specified in `core_env.tcl`. The target frequencies define the timing for various clock domains used by the processor core: **$core_clk_period**, **$bus_clk_period**.

The **$max_trans** TCL variable is used to specify the desired max transition rate (slew rate). The **$apr_margin** TCL variable reserves additional margins for the backend implementation; it is the amount of margin for place & route. The **$clock_uncertainty** TCL variable describes the expected clock uncertainty after clock tree synthesis. These two TCL variables are summed up together in the script (**$synthesis_margin** = **$apr_margin** + **$clock_uncertainty**) to set the synthesis tool's clock uncertainty value. Therefore, the actual cycle time for the synthesis will be (**$core_clk_period** - **$synthesis_margin**).

After the synthesis completes, the `output_netlist.tcl` script will reset the clock uncertainty to just **$clock_uncertainty** before writing out the SDC constraint file. However, if a shorter clock period is used for reserving margins for the backend implementation already, both **$clock_uncertainty** and **$apr_margin** should be set to 0 to avoid double counting.

Selecting Processor Configurations

The processor configuration is defined by the `config.inc` file that the configuration tool generates. The script `syn_env.tcl` automatically scans `config.inc` to determine and react to the selected configuration. The `config.inc` file should be properly saved to

$NDS_HOME/andes_ip/vc_core/top/hdl/config.inc

When executing the synthesis, `config.inc` will be copied to

$NDS_HOME/andes_ip/vc_core/syn/config.inc

for the synthesis script to find it.

Setting I/O Port Timing Constraints

The I/O delay constraints are set in `io_delay.tcl`. For bus signals, two thirds of the bus clock period is allocated to the external logic.

Setting Synthesis Environment

The synthesis environment setting in Table 148 could be configured with the following corresponding TCL script:

- syn_setup_dc.tcl

Table 148: Adjustable TCL Variables in NX25(F) Synthesis Scripts

Parameter	Description
tech_lib	Technology library name.
tech_lib_path	Path to the technology library.

Continued on next page...

Table 148: (continued)

Parameter	Description
operating_cond	Chip operating condition.
loading_cell	The input of library cell for output load estimation. For example, set loading_cell BUFX4.
driving_cell	The library cell for input driving estimation. For example, set driving_cell BUFX4.
dont_use_cells	The cells that should be excluded from the specified library during the synthesis.
wire_load_group	Wire load model selection group.
memory_lib_path	Path to memory library cells.
mem_cond	Memory macro file name suffix. Specify the file name suffix for searching the target memory library files in the memory path. The matched memory macro library file will be used for the synthesis.

Setting Synthesis Clock

The clock setting of the designs in Table 149 could be found in `core_env.tcl`.

Table 149: Adjustable TCL Variables in NX25(F) Synthesis Scripts

Parameter	Description
core_clk_period	CPU clock period in nanoseconds.
bus_clk_period	BUS clock period in nanoseconds.
test_clk_period	Test clock period in nanoseconds.
clock_uncertainty	Expected clock uncertainty in nanoseconds. The clock period will be deducted by (**$clock_uncertainty** + **$apr_margin**) for synthesis.
apr_margin	The timing margin for APR in nanoseconds. The clock period will be deducted by (**$clock_uncertainty** + **$apr_margin**) for the synthesis.

24.1.2.3 Reading Designs and Adding Memories

The script `read_design.tcl` is responsible for reading the NX25(F) RTL design. The script is located at

$NDS_HOME/andes_ip/vc_core/syn/script/read_design.tcl

In addition, the synthesizable definition of core memories (`vc_icache_ram.v`, `vc_dcache_ram.v`, `vc_btb_ram.v`, `vc_ilm_ram.v`, and `vc_dlm_ram.v`) should be created and saved under

$NDS_HOME/andes_ip/vc_core/memory/syn/

The SRAM cells for these memories should also be saved into the same directory and added to `read_design.tcl`.

The dimension of the used memories could be got by running `test_meminfo`. (See Section 23.3.8.)

24.1.3 Starting to Synthesize

Execute the run script, `run_syn`, under directory **$NDS_HOME**/andes_ip/vc_core/syn to start the synthesis. For example,

```
cd $NDS_HOME/andes_ip/vc_core/syn
./run_syn
```

24.1.4 Synthesis Result

24.1.4.1 Check Log File

Execute `check_log` to scan for synthesis error messages. The script must be run under **$NDS_HOME**/andes_ip/vc_core/syn. For example,

```
cd $NDS_HOME/andes_ip/vc_core/syn
./check_log
```

24.1.4.2 Check Report

After the synthesis completes, the timing and area reports are saved under directory **$NDS_HOME**/andes_ip/vc_core/syn/rpt. The final reports could be found in the files described below:

- Area report: area**${itr}**.rpt
- Timing summary report: timing_summary**${itr}**.rpt
- Detailed timing report: timing**${itr}**.rpt
- Power report: power**${itr}**.rpt

Where **${itr}** is the iteration number of incremental compiles.

24.1.4.3 Netlist, SDC, DB, and DDC Files

The netlist and SDC files are saved under the following directory:

$NDS_HOME/andes_ip/vc_core/syn/netlist

Note that if DC is in XG mode, the DDC file will also be saved in the directory below:

$NDS_HOME/andes_ip/vc_core/syn/ddc

24.2 Cadence Genus Synthesis

24.2.1 Introduction

The master run script that drives the entire synthesis flow is at

$NDS_HOME/andes_ip/vc_core/syn/run_syn_genus

And it is expected to be invoked under the `$NDS_HOME/andes_ip/vc_core/syn` directory. This script will set up working directories and invoke the synthesizer.

The Genus scripts and constraint files are under directory `$NDS_HOME/andes_ip/vc_core/syn/script_rc`. The top-level synthesis script is `vc_core.tcl`. It calls the rest of the synthesis scripts.

The tunable TCL variables that the synthesis scripts use are collected in the following files:

$NDS_HOME/andes_ip/vc_core/syn/core_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/script_rc/syn_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/syn_setup_genus.tcl

See Section 24.2.2 for more details. The only difference against Synopsys DC scripts is that there is no `output_netlist.tcl` for Cadence Genus, and the relevant code is directly inlined in `vc_core.tcl`.

After the synthesis completes, the results will be saved in the directories shown in the following table. These directories will be created if they do not exist.

Table 150: Synthesis Result Directories

Directory	Description
$NDS_HOME/andes_ip/vc_core/syn	Synthesis working directory
$NDS_HOME/andes_ip/vc_core/syn/log	Directory for log files
$NDS_HOME/andes_ip/vc_core/syn/netlist	Directory for netlist files
$NDS_HOME/andes_ip/vc_core/syn/rpt	Directory for synthesis reports

24.2.2 Synthesis Environment Setup

Table 151 and Table 152 show all TCL variables that can be adjusted, and they are discussed in the following subsections.

24.2.2.1 Technology Library and Memory Macros

Technology library and memory macros are specified in

$NDS_HOME/andes_ip/vc_core/syn/syn_setup_genus.tcl

In addition, this TCL file also contains settings of technology-related TCL variables, which include operating_cond, loading_cell, driving_cell, max_trans (max transitions), dont_use_cells, and wire_load_group (wire load model selection).

24.2.2.2 Synthesis Configuration

Four configuration scripts and a power pattern needed for performing synthesis are located separately at:

$NDS_HOME/andes_ip/vc_core/syn/core_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/pf_info.tcl
$NDS_HOME/andes_ip/vc_core/syn/verilog.tcf
$NDS_HOME/andes_ip/vc_core/syn/script_rc/syn_env.tcl
$NDS_HOME/andes_ip/vc_core/syn/script_rc/io_delay.tcl

The `core_env.tcl` configuration script sets TCL variables related to root design, target frequencies, and I/O timing. The `pf_info.tcl` script is for power report setting. The `verilog.tcf` is the power pattern. `pf_info.tcl` and `verilog.tcf` are generated automatically. The `syn_env.tcl` script parses `config.inc` for processor configurations. The `io_delay.tcl` script applies the actual I/O constraints.

Note that the `core_env.tcl` script is located one level higher than other scripts to be shared by all synthesis tools.

Setting the Target Frequencies
The target frequencies are specified in `core_env.tcl`. The target frequencies define the timing for various clock domains used by the processor core: **$core_clk_period**, **$bus_clk_period**.

The **$max_trans** TCL variable is used to specify the desired max transition rate (slew rate). The **$apr_margin** TCL variable reserves additional margins for the backend implementation; it is the amount of margin for place & route. The **$clock_uncertainty** TCL variable describes the expected clock uncertainty after clock tree synthesis. These two TCL variables are summed up together in the script (**$synthesis_margin** = **$apr_margin** + **$clock_uncertainty**) to set the synthesis tool's clock uncertainty value. Therefore, the actual cycle time for the synthesis will be (**$core_clk_period** - **$synthesis_margin**).

After the synthesis completes, the `vc_core.tcl` script will reset the clock uncertainty to just **$clock_uncertainty** before writing out the SDC constraint file. However, if a shorter clock period is used for reserving margins for the backend implementation already, both **$clock_uncertainty** and **$apr_margin** should be set to 0 to avoid double counting.

Selecting Processor Configurations

The processor configuration is defined by the `config.inc` file that the configuration tool generates. The script `syn_env.tcl` automatically scans `config.inc` to determine and react to the selected configuration. The `config.inc` file should be properly saved to

$NDS_HOME/andes_ip/vc_core/top/hdl/config.inc

When executing the synthesis, `config.inc` will be copied to

$NDS_HOME/andes_ip/vc_core/syn/config.inc

for the synthesis script to find it.

Setting I/O Port Timing Constraints

The I/O delay constraints are set in `io_delay.tcl`. For bus signals, two thirds of the bus clock period is allocated to the external logic.

Setting Synthesis Environment

The synthesis environment setting in Table 151 could be configured with the following corresponding TCL script:

- syn_setup_genus.tcl

Table 151: Adjustable TCL Variables in NX25(F) Synthesis Scripts

Parameter	Description
tech_lib	Technology library name.
tech_lib_path	Path to the technology library.
operating_cond	Chip operating condition.
loading_cell	The input of library cell for output load estimation. For example, set loading_cell BUFX4.

Continued on next page. . .

Table 151: (continued)

Parameter	Description
driving_cell	The library cell for input driving estimation. For example, set driving_cell BUFX4.
dont_use_cells	The cells that should be excluded from the specified library during the synthesis.
wire_load_group	Wire load model selection group.
memory_lib_path	Path to memory library cells.
mem_cond	Memory macro file name suffix. Specify the file name suffix for searching the target memory library files in the memory path. The matched memory macro library file will be used for the synthesis.

Setting Synthesis Clock

The clock setting of the designs in Table 152 could be found in `core_env.tcl`.

Table 152: Adjustable TCL Variables in NX25(F) Synthesis Scripts

Parameter	Description
core_clk_period	CPU clock period in nanoseconds.
bus_clk_period	BUS clock period in nanoseconds.
test_clk_period	Test clock period in nanoseconds.
clock_uncertainty	Expected clock uncertainty in nanoseconds. The clock period will be deducted by (**$clock_uncertainty** + **$apr_margin**) for synthesis.
apr_margin	The timing margin for APR in nanoseconds. The clock period will be deducted by (**$clock_uncertainty** + **$apr_margin**) for the synthesis.

24.2.2.3 Reading Designs and Adding Memories

The script `read_design.tcl` is responsible for reading the NX25(F) RTL design. The script is located at

$NDS_HOME/andes_ip/vc_core/syn/script_rc/read_design.tcl

In addition, the synthesizable definition of core memories (`vc_icache_ram.v`, `vc_dcache_ram.v`, `vc_btb_ram.v`, `vc_ilm_ram.v`, and `vc_dlm_ram.v`) should be created and saved under

$NDS_HOME/andes_ip/vc_core/memory/syn/

The SRAM cells for these memories should also be saved into the same directory and added to `read_design.tcl`.

The dimension of the used memories could be got by running `test_meminfo`. (See Section 23.3.8.)

24.2.3 Starting to Synthesize

Execute the run script, `run_syn_genus`, under directory **$NDS_HOME**/andes_ip/vc_core/syn to start the synthesis. For example,


```
cd $NDS_HOME/andes_ip/vc_core/syn

./run_syn_genus
```

24.2.4 Synthesis Result

24.2.4.1 Check Log File

Execute `check_log_genus` to scan for synthesis error messages. The script must be run under **$NDS_HOME**/andes_ip/vc_core/syn. For example,

```
cd $NDS_HOME/andes_ip/vc_core/syn

./check_log_genus
```

24.2.4.2 Check Report

After the synthesis completes, the timing and area reports are saved under directory **$NDS_HOME**/andes_ip/vc_core/syn/rpt. The final reports could be found in the files described below:

- Area report: area**${itr}**.rpt
- Timing summary report: timing_summary**${itr}**.rpt
- Detailed timing report: timing**${itr}**.rpt
- Power report: power**${itr}**.rpt

Where **${itr}** is the iteration number of incremental compiles.

24.2.4.3 Netlist, SDC, and DB Files

The netlist, SDC, and DB files are saved under the following directory:

$NDS_HOME/andes_ip/vc_core/syn/netlist

24.3 Timing Constraints

All NX25(F) timing constraints are collected in `timing_con.tcl` under the `script` or `script_rc` directory.

25 Synthesis of the Platform

25.1 Overview

This section discusses the reference synthesis flow for the accompanying AE350 platform. The reference flow is a bottom-up flow that requires synthesizing the component IPs first, before performing the synthesis of the platform itself.

25.2 Reference Scripts

The reference scripts include the following:

- Synthesis scripts (see Table 153) that are applicable to all platform IPs under:

  $NDS_HOME/andes_ip/peripheral_ip/design_flow/samples

- The file lists and constraint files for the synthesis of the chip-level platform module under:

  $NDS_HOME/andes_ip/ae350/syn

- The synthesizable definition of memory (ae350_rambrg_ram.v) should be created and saved under:

  $NDS_HOME/andes_ip/ae350/memory/syn

- A file list and a constraint file for the synthesis of each IP under:

  $NDS_HOME**/andes_ip/peripheral_ip/**$IP_NAME/syn

Table 153: Reference Synthesis Scripts

File Name	Description
ip_env.tcl	Providing TCL variables used by the reference scripts for the synthesis of the chip-level module of the selected platform
Cadence Genus	
run_genus.sh	Main script to drive syntheses of all peripheral IPs and the platform for Genus
syn_genus.tcl	Top script for synthesizing one component IP
syn_setup_genus.tcl	Synthesis environment setup
syn_run_genus.tcl	Main synthesis script
Synopsys DC	

Continued on next page...

Table 153: (continued)

File Name	Description
run_dc.sh	Main script to drive syntheses of all peripheral IPs and the platform for DC
syn_dc.tcl	Top script for synthesizing one component IP
syn_setup_dc.tcl	Synthesis environment setup
syn_run_dc.tcl	Main synthesis script

25.3 Setting Environment Variables and TCL Variables

The environment variables listed in Table 154 are used by the reference scripts and must be set properly before invoking the synthesis command.

Table 154: Variables for Synthesis

Variable Name	Description
OS environment variable	
SCRIPT_PATH	Path of the synthesis scripts (**$NDS_HOME**/andes_ip/peripheral_ip/design_flow/samples)
TCL variables in ip_env.tcl	
env(core_clk_period)	Period of the NX25(F) clock
env(aclk_period)	Period of the AXI clock
env(hclk_period)	Period of the AHB clock
env(pclk_period)	Period of the APB clock
env(jdtm_clk_period)	Period of clock for the external debugger interface (NCEJDTM200)
env(pclk_period)	Period of the APB clock
env(osch_clk_period)	Period of the main clock for chip
env(spi_clk_period)	Period of the SPI clock
env(ip_def_search_path)	Search path for include files
syn_define	Macro definitions for synthesis
compile_itr	Number of synthesis iterations
report_path	Path of reports
output_path	Path of the output netlists/constraints
ip_database	Path to all netlists/constraints of component IPs for the synthesis of the chip-level module of the selected platform.

Continued on next page...

Table 154: (continued)

Variable Name	Description
TCL variables in `syn_setup_XXX.tcl`(See Section 24.1.2 for more information)	
tech_lib	Name of the standard cell library
tech_lib_path	Path of the standard cell library
operating_cond	Operating condition of the standard cell library
mem_lib	Name of the memory library
memory_lib_path	Path of the memory library
mem_cond	Operating condition of the memory library
pad_lib	Name of the PAD library
pad_lib_path	Path of the PAD library
loading_cell	Loading cell name
driving_cell	Driving cell name
dont_use_cells	List of cells which should not be used
wire_load_group	Name of the wire-load selection group
syn_effort	Synthesis effort

25.4 Batch Script

A reference batch script is available for driving the whole chip synthesis in one shot, including NX25(F) processor, all peripheral IPs and the chip-level of the platform. These scripts contain the **$itr** variable, which assigns the iteration of NX25(F) processor synthesis result for copying and re-naming the necessary netlist files to the **$ip_database** directory. The **$itr** variable can be revised based on the requirement (The default itr variable value is 2).

- For Synopsys DC

  $SCRIPT_PATH/run_dc.sh

- For Cadence Genus

  $SCRIPT_PATH/run_genus.sh

25.5 Synthesizing the NX25(F) Processor

The synthesis result of the selected NX25(F) processor must be ready before the synthesis of the chip-level module of the platform. Please see Section 24 for more information. The following table lists the necessary files which are copied and renamed from the best synthesis iteration result in the **$ip_database** directory for the synthesis of the platform.

File Name	Source Directory	Applied EDA Tool
AE350 Platform		
ae350_cpu_subsystem.ddc	**$NDS_HOME**/andes_ip/vc_core/syn/ddc	Synopsys DC
ae350_cpu_subsystem.vg	**$NDS_HOME**/andes_ip/vc_core/syn/netlist	Cadence Genus
ae350_cpu_subsystem.sdc	**$NDS_HOME**/andes_ip/vc_core/syn/netlist	Cadence Genus

25.6 Synthesizing Peripheral IPs

The synthesis result of the peripheral IPs must also be ready before the synthesis of the chip-level module of the platform. To synthesize a peripheral IP, environment variable **$DESIGN_NAME** should be set to the IP name in the lower case. For example, the following command sets the environment variable for the GPIO controller, ATCGPIO100:

- For Bourne shell:

```
DESIGN_NAME=atcgpio100; export DESIGN_NAME
```

- For C shell:

```
setenv DESIGN_NAME atcgpio100
```

Each IP should be synthesized under its own working directory, by creating directories as follows:

```
mkdir $DESIGN_NAME
cd $DESIGN_NAME
```

Under the working directory, start the synthesis with the following command:

- For Cadence Genus

```
genus -f $SCRIPT_PATH/syn_genus.tcl -log ./log/genus.log
```

- For Synopsys DC

```
dc_shell-t -f $SCRIPT_PATH/syn_dc.tcl | tee dc.log
```

When the synthesis completes successfully, the synthesis report will be generated in the directory **$report_path**. The netlist, SDC file and DDC file will be generated in the directory **$output_path** and copied to the **$ip_database** directory.

25.7 Synthesizing the Chip-Level Module of the Platform

When the syntheses of the NX25(F) processor and peripheral IPs are complete, the chip-level of the platform can synthesized as follows:

- Set environment variable **$DESIGN_NAME** to the chip-level module name of the selected platform, i.e., ae350_chip.
- Follow the same procedures as described in Section 25.6 for the synthesis.

26 FPGA

26.1 FPGA Block Diagram

The AE350 modules and the external components on the EVB are illustrated in Figure 31. AE350 interfaces with external components by two UART ports, two SPI ports, up to 4 PWM channels, 32 bits GPIO, I2C, and a JTAG debug port.

26.1.1 UART

UART1 is a reserved port; UART2 is connected to the UART DB9 male connector for connecting to terminal emulators.

26.1.2 JTAG Debug Port

The JTAG debug port is connected to the JTAG header for communicating with the AICE-MICRO probe.

26.1.3 SPI

SPI1 is connected to the on-board flash ROM. SPI2 is connected to the connector pins shown in Table 158. Another SPI ROM could be connected to SPI2 through these pins.

26.1.4 PWM

Four PWM channels are connected to the connector pins shown in Table 162.

26.1.5 GPIO

Two seven-segment LEDs are connected to part of the GPIOs for displaying diagnostic codes during booting or GPIO testing. The rest of GPIOs are connected to the buttons and the on-board connector pins. See Table 163 for pin assignments.

26.1.6 I2C

The I2C port is connected to an on-board I2C ROM.

26.1.7 Clock Generator

The oscillators on the ADP-XC7KFF676 EVB generate a 20MHz clock source and a 32.768KHz clock source. The clock generator produces the following clocks by default:

- CPU clock (60 MHz)
- AXI clock (60 MHz)
- AHB clock (60 MHz)
- APB clock (60 MHz)
- UART clock (20 MHz)
- SPI clock (66 MHz)
- RTC 32K clock (32.768 KHz)

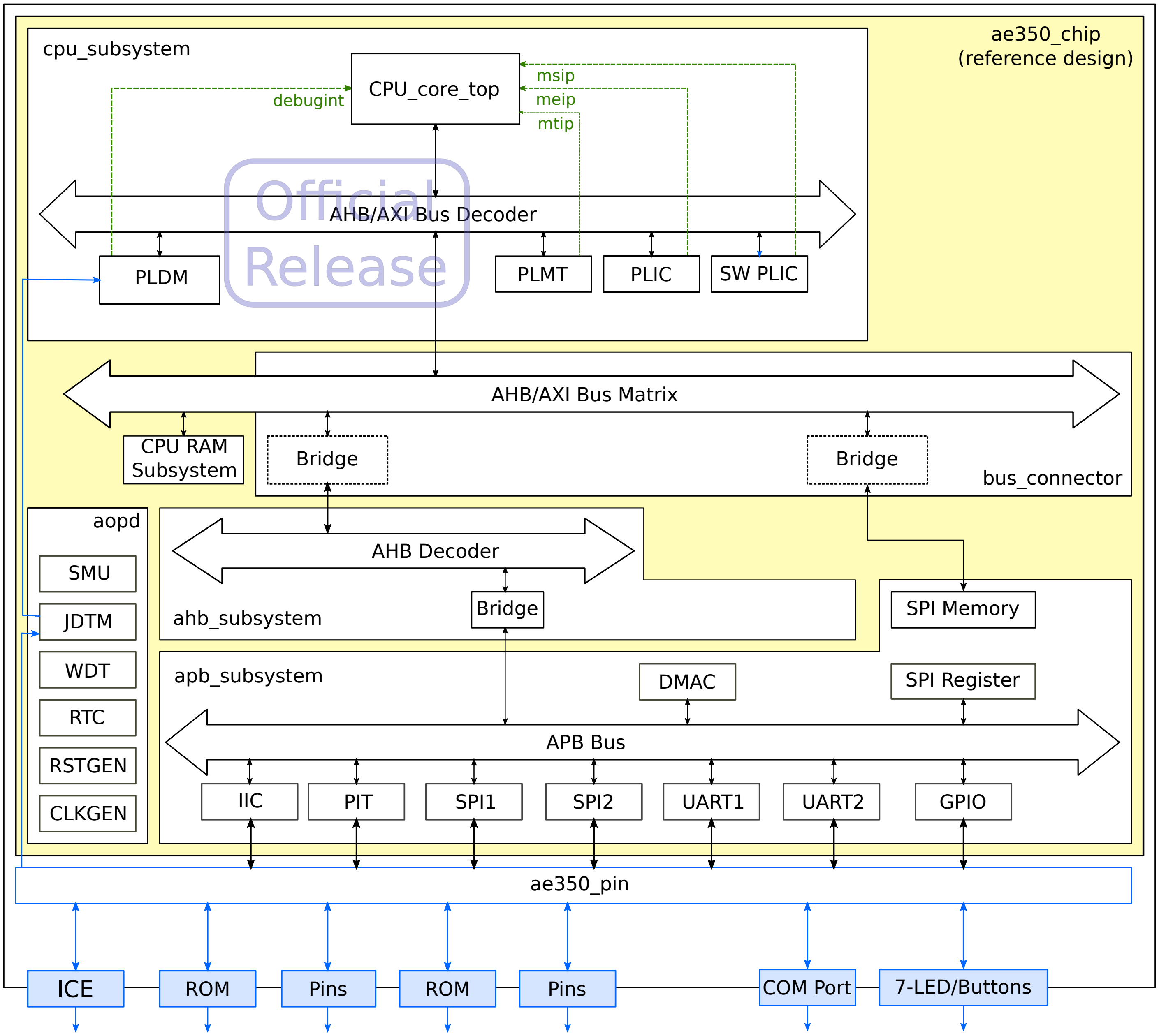


Figure 31: AE350 FPGA Block Diagram

26.2 FPGA Pin Assignment

26.2.1 Global Signals

Table 155: Pin Assignment of Global Signals

Signal Name	FPGA Pin #	Board Pin Name	Component	Remark
X_hw_rstn	U22	SYS_RSTn	HW_RST_SW1	
X_oschin	G24	OSCCLK1	X1	
X_osclin	H17	RTC_32K	X2	
X_wakeup_in	D9	GPIO8	SW8	
X_por_b	U21	PORn	ALIVE Power On Reset	
X_aopd_por_b	-	-	-	Internally connected to X_por_b
X_om	-	-	-	Internally hardwired to 0
X_oschio	-	-	-	Not used on FPGA
X_osclio	-	-	-	Not used on FPGA
X_rtc_wakeup	-	-	-	Not used on FPGA
X_mpd_pwr_off	-	-	-	Not used on FPGA

26.2.2 JTAG Signals

The JTAG signals are for connecting to the external debugger.

Table 156: Pin Assignment of JTAG Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_tck	Y22	ICE_TCK	AICE_CON1.9	
X_tms	AA24	ICE_TMS	AICE_CON1.7	
X_tdo	AA22	ICE_TDO	AICE_CON1.13	
X_tdi	AC23	ICE_TDI	AICE_CON1.5	
X_trst	W20	ICE_TRSTn	AICE_CON1.3	

26.2.3 SPI 1: For Flash ROM

Table 157: Pin Assignment of SPI1 Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_spi1_mosi	AC6	SPI_SI	SPI ROM U17.5	
X_spi1_miso	AC4	SPI_SO_SIO1	SPI ROM U17.2	
X_spi1_clk	AA5	SPI_SCLK	SPI ROM U17.6	
X_spi1_csn	Y5	SPI_CSN	SPI ROM U17.1	
X_spi1_holdn	AB6	SPI_SIO3	SPI ROM U17.7	
X_spi1_wpn	AB5	SPI_WP#_SIO2	SPI ROM U17.3	

26.2.4 SPI 2

Table 158: Pin Assignment of SPI2 Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_spi2_mosi	H14	CFC_ADDR2	IDE_CON1.36	
X_spi2_miso	C14	CFC_NCE1	IDE_CON1.38	
X_spi2_clk	J11	CFC_ADDR0	IDE_CON1.35	
X_spi2_csn	E12	CFC_NCE0	IDE_CON1.37	
X_spi2_holdn	J10	CFC_ADDR1	IDE_CON1.33	
X_spi2_wpn	H12	CFC_PDIAG	IDE_CON1.34	

26.2.5 UART1 & UART2

Table 159: Pin Assignment of UART1 Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_uart1_rxd	G25	S2_RXD	RS-232 U15.17	
X_uart1_txd	D25	S2_TXD	RS-232 U15.12	
X_uart1_ctsn		-	-	
X_uart1_rtsn		-	-	
X_uart1_dcdn		-	-	Not used on FPGA
X_uart1_dsrn		-	-	Not used on FPGA
X_uart1_dtrn		-	-	Not used on FPGA
X_uart1_out1n		-	-	Not used on FPGA
X_uart1_out2n		-	-	Not used on FPGA
X_uart1_rin		-	-	Not used on FPGA

Table 160: Pin Assignment of UART2 Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_uart2_rxd	R18	S1_RXD	RS-232 U15.19	
X_uart2_txd	T17	S1_TXD	RS-232 U15.14	
X_uart2_ctsn		-	-	
X_uart2_rtsn		-	-	
X_uart2_dcdn		-	-	Not used on FPGA
X_uart2_dsrn		-	-	Not used on FPGA
X_uart2_dtrn		-	-	Not used on FPGA
X_uart2_out1n		-	-	Not used on FPGA
X_uart2_out2n		-	-	Not used on FPGA
X_uart2_rin		-	-	Not used on FPGA

26.2.6 I2C

Table 161: Pin Assignment of I2C Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_i2c_scl	M25	I2C_SCL	I2C FLASH U16.6	
X_i2c_sda	L25	I2C_SDA	I2C FLASH U16.5	

26.2.7 PWM

Table 162: Pin Assignment of PWM Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_pwm_ch0	A15	CFC_RESET	IDE_CON1.1	
X_pwm_ch1	C12	CFC_DATA7	IDE_CON1.3	
X_pwm_ch2	D10	CFC_DATA6	IDE_CON1.5	
X_pwm_ch3	E10	CFC_DATA5	IDE_CON1.7	

26.2.8 GPIO

Table 163: Pin Assignment of GPIO Signals

Signal Name	FPGA Pin #	Board Pin Name	Component Pin	Remark
X_gpio[0]	E21	GPIO1	SW1	
X_gpio[1]	F24	GPIO2	SW2	
X_gpio[2]	J21	GPIO3	SW3	
X_gpio[3]	L23	GPIO4	SW4	
X_gpio[4]	A13	GPIO5	SW5	
X_gpio[5]	A12	GPIO6	SW6	
X_gpio[6]	J14	GPIO7	SW7	
X_gpio[7]	C11	CFC_DATA8	IDE_CON1.4	
X_gpio[8]	E11	CFC_DATA9	IDE_CON1.6	
X_gpio[9]	D11	CFC_DATA10	IDE_CON1.8	
X_gpio[10]	F14	CFC_DATA11	IDE_CON1.10	
X_gpio[11]	F13	CFC_DATA12	IDE_CON1.12	
X_gpio[12]	G12	CFC_DATA13	IDE_CON1.14	
X_gpio[13]	F12	CFC_DATA14	IDE_CON1.16	
X_gpio[14]	D14	CFC_DATA15	IDE_CON1.18	
X_gpio[15]	J8	CFC_DATA4	IDE_CON1.9	
X_gpio[16]	V26	7SEG1_A	7SEG1.A	
X_gpio[17]	W25	7SEG1_B	7SEG1.B	
X_gpio[18]	W26	7SEG1_C	7SEG1.C	
X_gpio[19]	V21	7SEG1_D	7SEG1.D	
X_gpio[20]	W21	7SEG1_E	7SEG1.E	
X_gpio[21]	AA25	7SEG1_F	7SEG1.F	
X_gpio[22]	AB25	7SEG1_G	7SEG1.G	
X_gpio[23]	W23	7SEG1_P	7SEG1.P	
X_gpio[24]	W24	7SEG2_A	7SEG2.A	
X_gpio[25]	AB26	7SEG2_B	7SEG2.B	
X_gpio[26]	AC26	7SEG2_C	7SEG2.C	
X_gpio[27]	Y25	7SEG2_D	7SEG2.D	
X_gpio[28]	Y26	7SEG2_E	7SEG2.E	
X_gpio[29]	AD21	7SEG2_F	7SEG2.F	
X_gpio[30]	AD23	7SEG2_G	7SEG2.G	
X_gpio[31]	AB24	7SEG2_P	7SEG2.P	

26.3 IO Constraints

The chip-level IO pins of the NX25(F) platform consist of pins mainly from peripheral controllers (e.g., SPI, UART) which communicate with off-chip components. Apart from them, the RISC-V debug transport module NCEJDTM200 also requires some IO pins for interfacing with the external debugger. As for NX25(F), all its interface signals are directly connected to other on-chip components.

This section only describes the IO constraints for the external debug interface of NCEJDTM200. Constraints related to other peripheral IPs can be found in respective data sheets of those peripheral IPs.

26.3.1 IO Constraints for the External Debug Interface

It is expected that the external debug interface should normally work with frequency no higher than 25MHz, so the FPGA I/O constraint for this interface could be set as follows:

```
create_clock -name {X_tck} [get_ports {X_tck}] -period 40.0 -waveform {0 20.0}
set_clock_groups -asynchronous -name {X_tck_async_SDC} -group [get_clocks {X_tck}]

set_output_delay   9.0 [get_ports {X_tdo}] -clock {X_tck} -add_delay
set_input_delay   30.0 [get_ports {X_tdi}] -clock {X_tck} -add_delay
set_output_delay   9.0 [get_ports {X_tms}] -clock {X_tck} -add_delay
set_input_delay   30.0 [get_ports {X_tms}] -clock {X_tck} -add_delay
```

26.3.2 IO Constraints Except the External Debug Interface

For other I/O constraints, please refer to the following constraint files.

- For AE350 Platform

  $NDS_HOME/andes_ip/ae350/fpga/vivado_flow/constraint/ae350_fpga_orca_pre_synth.sdc
  $NDS_HOME/andes_ip/ae350/fpga/vivado_flow/constraint/ae350_fpga_orca_post_synth.sdc

26.4 FPGA Netlist Generation

This section describes FPGA synthesis flow to create the bitmap for the NX25(F) platform.

The FPGA synthesis flow requires the Xilinx Vivado Design Suite and it generates the bitmap for the AndeShape ADP-XC7KFF676 evaluation board.

The FPGA synthesis environment and the working directory are at:

$NDS_HOME/andes_ip/ae350/fpga

26.4.1 FPGA Macros Generation

Several FPGA memory macros and DCM macros are required for the synthesis of the AE350 platform. These macros are not part of the NX25(F) platform so they need to be generated separately using Xilinx tools. A script file is included to generate all the required macros automatically:

- For AE350 Platform

```
cd $NDS_HOME/andes_ip/ae350/fpga
./gen_fpga_lib clk mem ila -part xc7k160t
```

The generated macros and the file list for FPGA synthesis will be saved under the following directory:

$NDS_HOME/vendor_ip/xilinx_ip/xc7k410tffg676-2

26.4.2 FPGA Synthesis

Invoke the following commands to start the FPGA synthesis:

- For AE350 Platform

```
cd $NDS_HOME/andes_ip/ae350/fpga
./syn_fpga_vivado -part xc7k160t
```

26.4.3 FPGA Synthesis Result

The synthesis results will be saved in the following folders of the working directory:

- fpga_ae350_vivado

  Xilinx FPGA BIN file (ae350_chip.bin)
  Xilinx FPGA MCS file (ae350_chip.mcs)

- fpga_ae350_vivado/ae350_chip

  Xilinx FPGA BIT file (ae350_chip.bit)

27 DFT and MBIST

The NX25(F) design does not include DFT/MBIST logic circuit. It is up to the implementation to decide the most suitable testing strategy.