RISC-V MC CPU IP – Lattice Propel Builder 2025.1

IP Version: v2.8.0

User Guide

FPGA-IPUG-02278-1.1

July 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Contents

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Tables

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AHB-Lite	Advanced High-performance Bus – Lite
AMO	Atomic Memory Operation
CX	Composable Extension
CXU	Composable Extension Unit
CXU-LI	Composable Extension Unit Logic Interface
CF	Custom Function
CFU	Custom Function Unit
CFU-LI	Custom Function Unit Logic Interface
CI	Custom Interface
CPU	Central Processing Unit
CSR	Control and Status Register
DMIPS	Dhrystone MIPS (Million Instructions per Second)
FPGA	Field Programmable Gate Array
GDB	Gnu Debugger
HDL	Hardware Description Language
IP	Intellectual Property
IRQ	Interrupt Request
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
LUT	Lookup-Table
MC	Micro-Controller (RISC-V for Micro-Controller applications)
OpenOCD	Open On-Chip Debugger
PIC	Programmable Interrupt Controller
RISC-V	Reduced instruction set computer-V (five)
RVFI	RISC-V Formal Interface
RV32IMC	RISC-V Integer, M and Compressed Instruction Sets
WFI	Wait For Interrupt

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1. Introduction

The Lattice Semiconductor RISC-V MC CPU soft IP contains a 32-bit RISC-V processor core and optional submodules – Timer and Programmable Interrupt Controller (PIC). The CPU core has optional instruction and data caches. The CPU core supports the RV32IMCE instruction set, external interrupts, and debug feature that is JTAG – IEEE 1149.1 compliant. The Timer submodule is a 64-bit real-time counter, which compares a real-time register with another register to assert the timer interrupt. The PIC submodule aggregates up to eight external interrupt inputs into one interrupt to the CPU core. The submodule registers are accessed by the processor core using a 32-bit Advanced High-performance Bus – Lite (AHB-Lite) interface.

The design is implemented using Verilog HDL, and it can be configured and generated using the Lattice Propel™ Builder software. It supports Certus™-N2, Lattice Avant™, MachXO5™-NX, CrossLinkU™-NX, CrossLink™-NX, CertusPro™-NX, Certus-NX, MachXO3D™, MachXO3™, and MachXO2™ FPGA devices.

1.1. What's New in This IP Release

- Added the *Enable RVFI* attribute to support RISC-V Formal Interface (RVFI).
- Changed the attribute name *Number of Interrupt Requests* to *Number of PIC Interrupt Requests Input* and updated its minimal value from *2* to *0*.
- Added the *Reset Vector* attribute for setting the value of the reset vector.
- Added the *E Extension for Embedded Device* attribute.
- Added the *Enable AHBL Data Output Register* attribute to support optional register slice on Data Port with cache disabled.

1.2. Quick Facts

Table 1.1 presents a summary of the RISC-V MC CPU IP.

Table 1.1. RISC-V MC CPU IP Quick Facts

IP Requirements	Supported FPGA Family	Certus-N2, Lattice Avant, MachXO5-NX, CrossLinkU-NX, CrossLink-NX, CertusPro-NX, Certus-NX, MachXO3D, MachXO3L™, MachXO3LF™, MachXO2
Resource Utilization	Supported User Interfaces	AHB-Lite Interface, Composable Extension Unit Logic Interface (CXU-LI), RISC-V Formal Interface (RVFI)
	Resources	See Table A.1, Table A.2, and Table A.3.
Design Tool Support	Lattice Implementation	IP v2.8.0 – Lattice Propel Builder 2025.1, Lattice Radiant™ 2025.1
	Simulation	For a list of supported simulators, see the Lattice Radiant and Lattice Diamond™ software user guide.

1.3. Features

The RISC-V MC soft IP has the following features:

- RV32IMCE instruction set
- Five stage pipeline
- Supports the AHB-Lite bus standard for instruction and data ports.
- Supports CXU-LI
- Supports RVFI
- Supports dynamic branch target prediction.
- Optional caches including a 4 KB two-way instruction cache and a 4 KB two-way data cache, for Certus-N2, Lattice Avant, MachXO5-NX, Certus-NX, CertusPro-NX, and CrossLink-NX devices only
- Optional debug using Gnu Debugger (GDB) and Open On-Chip Debugger (OpenOCD)
- Optional PIC module
- Optional Timer module
- Interrupt and exception handling under Machine Mode

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

- Supports the DMIPS performance listed in Table 1.2.
- Fmax > 100 MHz on CrossLink-NX devices, tested on the Hello World template provided by Lattice Propel design environment.
- Configurable reset vector

Table 1.2. DMIPS Performance

RISC-V Core	RISC-V Configuration	DMIPS
MC	RV32IMCE, no cache	0.45 DMIPS/MHz
MC	RV32IMCE, with ID cache	1.25 DMIPS/MHz

Note: the system memory in the SoC used to measure DMIPS has an output register.

1.4. Conventions

1.4.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.4.2. Signal Names

- _n are active low, asserted when value is logic 0.
- _i are input signals.
- _o are output signals.
- _io are bidirectional signals.

1.5. Licensing and Ordering Information

The MC CPU IP is provided at no additional cost with the Lattice Propel design environment. The IP can be fully evaluated in hardware without requiring an IP license string.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. Functional Descriptions

2.1. Overview

The RISC-V MC CPU IP processes data and instructions while monitoring external interrupts. As shown in Figure 2.1, the CPU IP has a 32-bit processor core and optional submodules. It uses a read-only AHB-Lite interface for instruction fetch and another AHB-Lite interface with read/write access for data access. See Table 2.5 and Table 2.6 for the AHB-Lite Instruction Fetch and Data Accessing ports definition. The CPU core, PIC, Timer, and AHB-Lite multiplexor run in the system clock domain. The Core Debug runs in both system clock domain and JTAG clock domain.

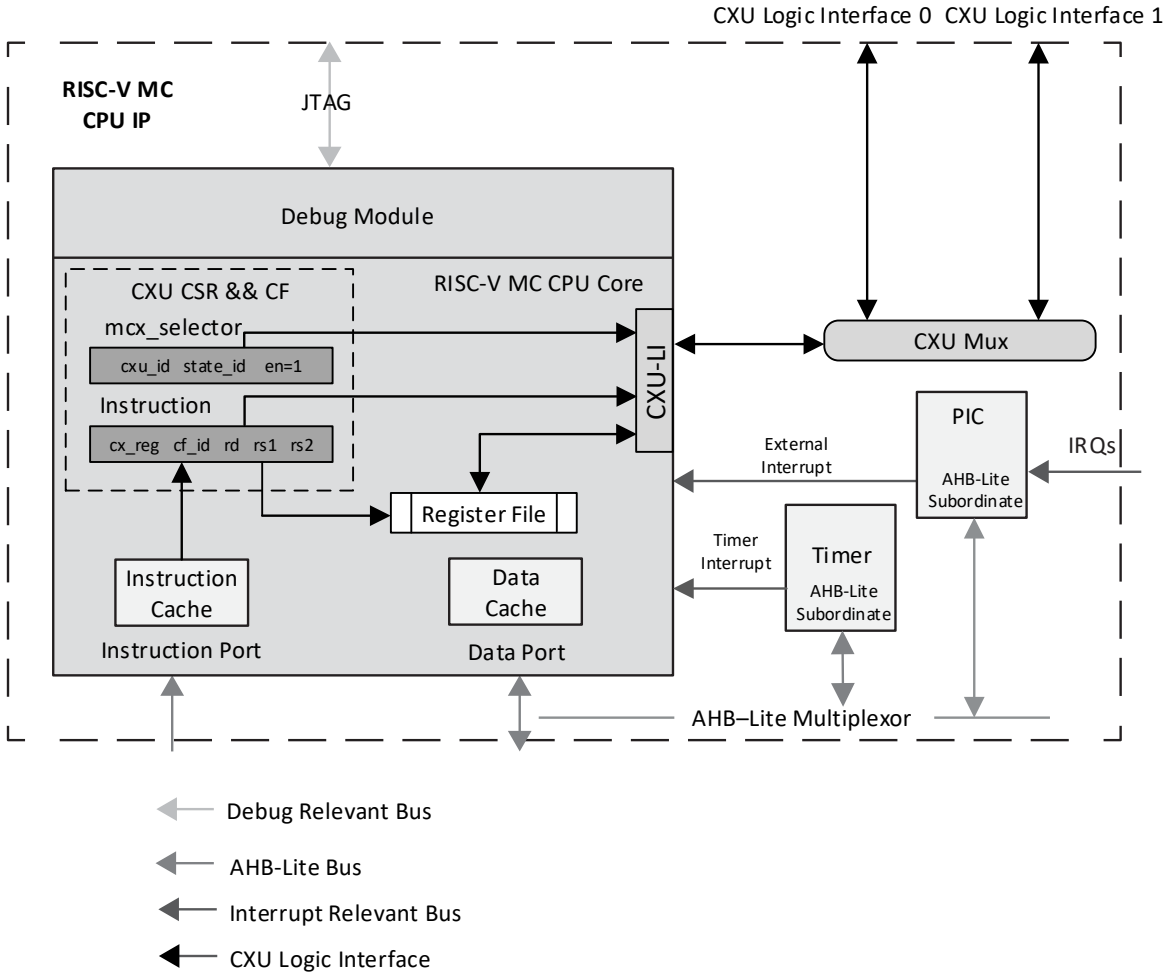

Figure 2.1. RISC-V MC Soft IP Diagram

2.2. Modules Description

2.2.1. RISC-V Processor Core

The processor core follows the RV32IMCE instruction set and the M, C, and E extensions are optional in Module/IP Block Wizard GUI General Tab, as shown in Figure 2.3. Figure 2.2 shows the processor core block diagram.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

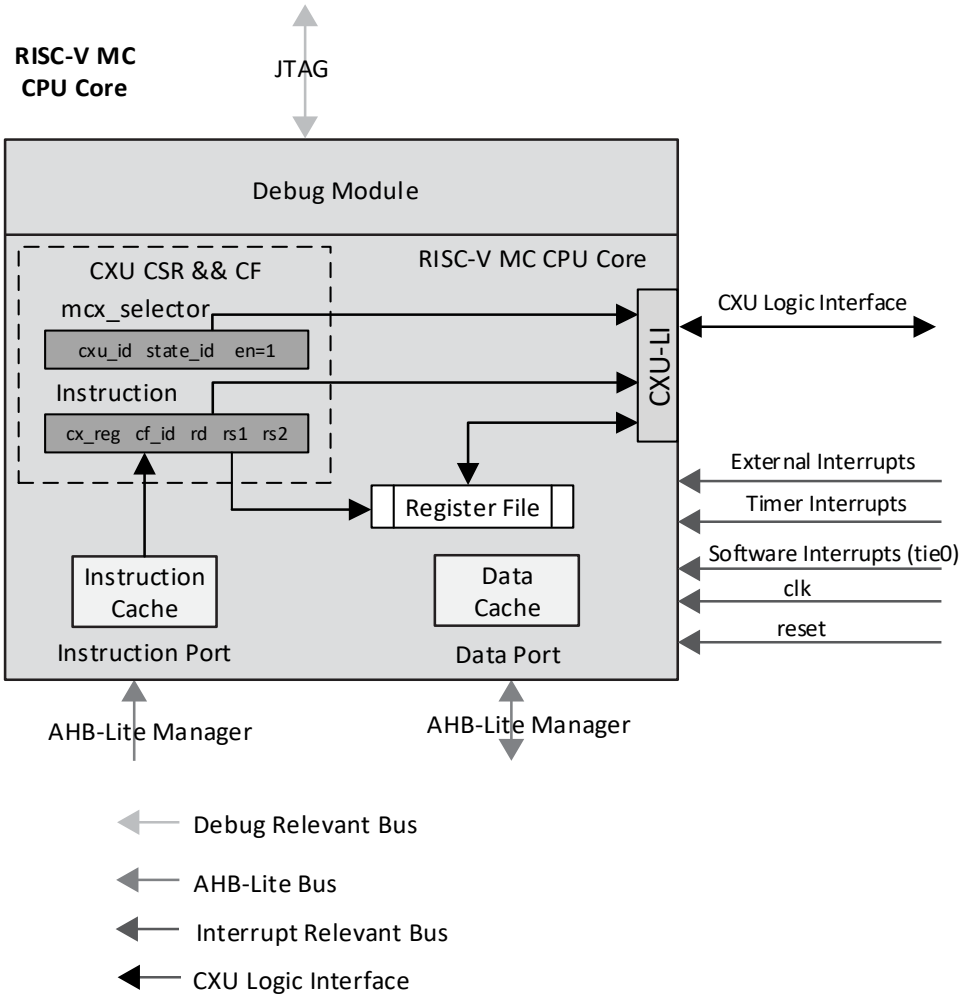

Figure 2.2. RISC-V MC Processor Core Block Diagram

2.2.1.1. Reset Vector

The reset vector of the processor is configurable through the Reset Vector attribute, which is under the General tab in the Module/IP Block Wizard GUI (Figure 2.3).

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

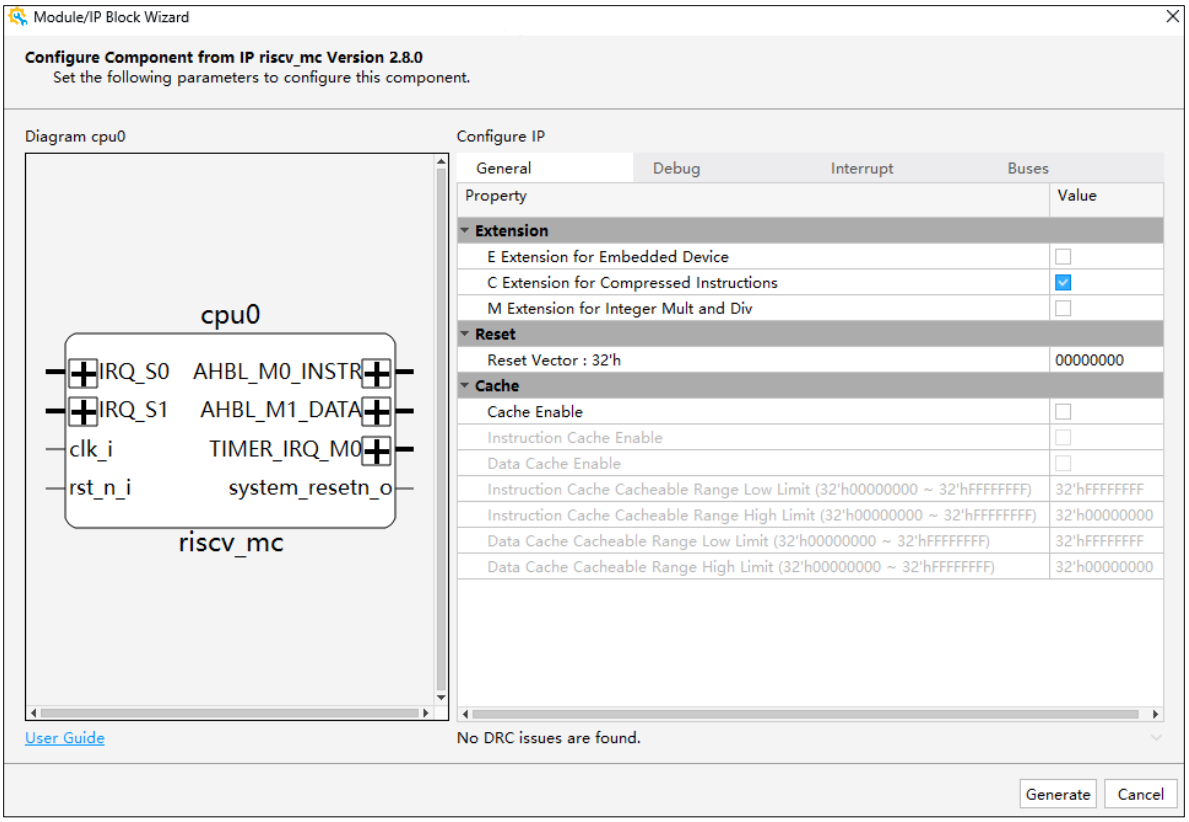

Figure 2.3. General Tab

2.2.1.2. Instruction and Data Caches

The processor core supports optional instruction and data caches.

The instruction and data caches are both 4 KB two-way set associative, and each cache line contains 32 bytes. The cache strategy for data cache is write through, and the cache eviction policy of both caches is round robin.

As shown in Figure 2.3, the instruction and data caches can be enabled or disabled together by checking or unchecking the Cache Enable option when instantiating the CPU in the Lattice Propel design environment, and the cacheable address range can be configured by setting the Instruction/Data Cache Cacheable Address Range Low Limit and Instruction/Data Cache Cacheable Address Range High Limit parameters according to application demands.

To flush the caches, refer to annotations of cache.h in the driver codes. The cache invalidates the corresponding cache line and reloads it from memory the next time it is accessed. Those instructions are accepted only if the cache is enabled. If the cache is not enabled, those instructions raise an illegal instruction exception.

It should be noted that the instruction and data caches should only be enabled for Certus-N2, Lattice Avant, MachXO5-NX, CrossLink-NX, CertusPro-NX, and Certus-NX devices, as they have enough resources to support the caches.

2.2.1.3. Branch Prediction

Processors with caches use dynamic target prediction for branches and processors without caches do not implement branch prediction.

2.2.1.4. Debug

The processor core supports the IEEE-1149.1 JTAG debug logic with two hardware breakpoints.

To use the debug module, it is required to allow writes from the data port to the instruction memory in the SoC. Single-port instruction memory is not allowed to debug.

The JTAG channel is configurable in the Module/IP Block Wizard GUI, as shown in Figure 2.4. The channel range depends on the kind of FPGA device family. For Certus-N2 and Lattice Avant and Nexus family devices, the default JTAG

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

channel range is 14–16. When enabling the Extend JTAG Channel attribute, the channel range enlarges from 14–16 to 14–24 for Certus-N2 and Lattice Avant family devices. For Nexus family devices, the channel range enlarges from 14–16 to 10–18. The soft JTAG is available on Certus-N2, Lattice Avant, MachXO5-NX, CrossLink-NX, CertusPro-NX, and Certus-NX devices. For more information, refer to Appendix B.

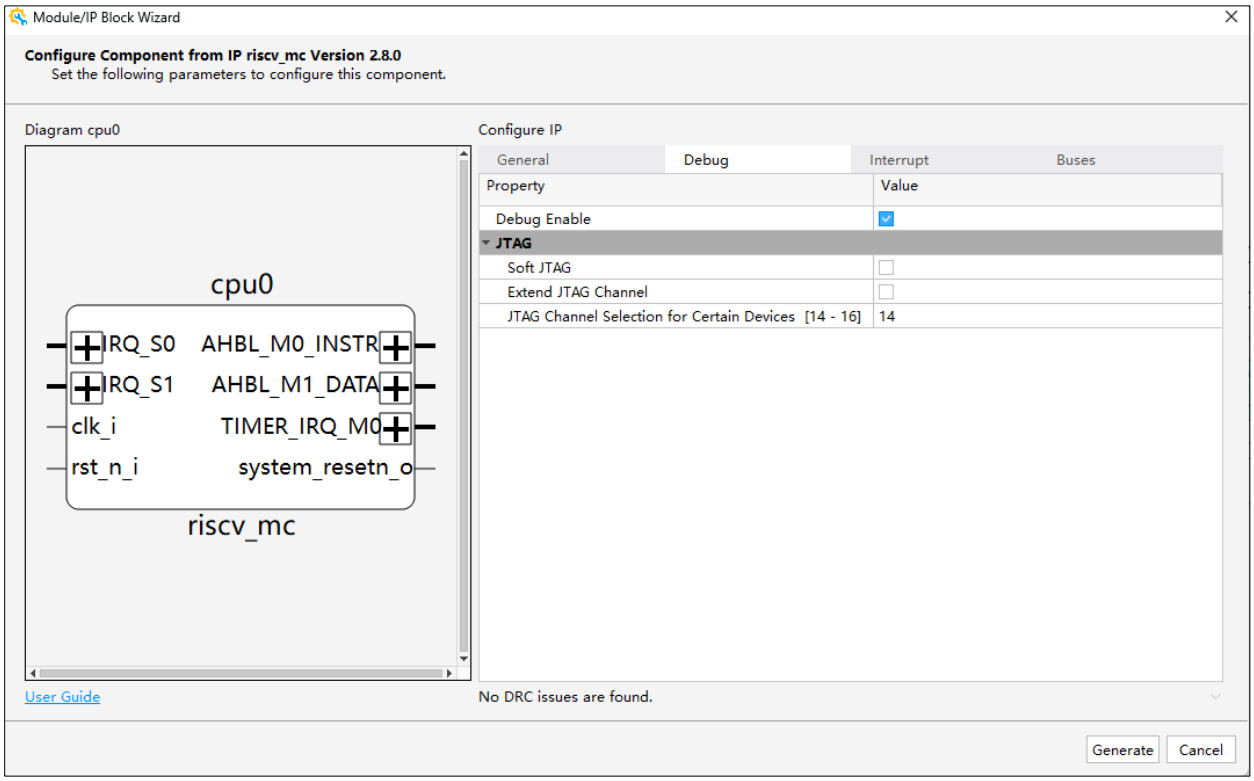

Figure 2.4. Debug configuration Tab

2.2.1.5. Interrupt

The core CPU's interrupts are level sensitive and high active. A given interrupt should remain asserted until cleared by the corresponding interrupt service routine.

2.2.1.6. Exception

If an exception occurs, the processor core stops the corresponding instruction. It flushes the exception instruction and instructions in the pipeline fetched after the exception. Then, the core waits until all the flushed instructions reach the writeback stage before jumping to the exception service routine.
Note: In the firmware, the exception handler return address is fixed at mepc + 4. For the C compressed code, the exception recovery is not guaranteed.

Starting from version 2.7.0, the MC core supports the write response. A write error on the AHB-L data bus causes the store access exception with ID 7.

2.2.1.7. WFI for Low Power

The processor core enters into low power mode with Wait For Interrupt (WFI) instruction. The program counter halts during low power mode, and the processor wakes up if there is an external or timer interrupt.

2.2.1.8. Control and Status Registers

The processor core supports the Control and Status Registers (CSRs) listed in Table 2.1.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.1. RISC-V Processor Core Control and Status Registers

CSR No.	CSR Name	Access	Fields
0x300	Machine Status (mstatus)	read/write	bit[12:11]: mpp, privilege mode before entering a trap, should always be 2'b11 in machine mode in this CPU core. bit[7]: mpie, mie before entering a trap, updates to mie value when entering a trap. bit[3]: mie, global interrupt enable.
0x301	Machine ISA (misa)	read-only	bit[31:30]: base, hardwired 0x1, stands for RV32. bit[25:0]: extension, stands for the supported ISAs.
0x304	Machine Interrupt Enable (mie)	read/write	bit[11]: meie, machine mode external interrupt enable. bit[7]: mtie, machine mode timer interrupt enable. bit[3]: msie, machine mode software interrupt enable.
0x305	Machine Trap-Vector Base-Address (mtvec)	read/write initialized to 0x20	bit[31:2]: trap vector base address, 4-byte aligned. bit[0]: trap vector mode, all traps set the program counter to the base address in the RISC-V MC CPU core. Bit[1] is not supported. Only 1'0 – direct mode and 1'b1 – vectored mode are available.
0x340	Machine Scratch (mscratch)	read/write	bit[31:0]: in machine mode, it is used to hold a pointer to a machine-mode hart-local context space and is swapped with a user register upon entry to a machine mode trap handler.
0x341	Machine Exception Program Counter (mepc)	read/write	bit[31:0]: when a trap is taken into machine mode, mepc is used to store the address of the instruction that encounters the exception.
0x342	Machine Cause (mcause)	read-only	bit[31]: 1'b1 – interrupt, 1'b0 – exception bit[3:0]: exception code for interrupt: • 3 – machine software interrupt • 7 – machine timer interrupt • 11 – machine external interrupt for exception : • 0 – instruction address misaligned • 1 – instruction access fault • 2 – illegal instruction • 4 – load address misaligned • 5 – load access fault • 7 – write access fault
0x343	Machine Trap Value (mtval)	read-only	mtval stores the bad address when an exception occurs in machine mode. The source of bad address can be: • exceptions on dbus, such as access error, mmu exception, or unaligned access. In this condition, it contains dbus address. • exceptions from CSRs, such as ecall and ebreak. In this condition, it contains the execute instruction. • exceptions by fetch. It contains fetch pc. • exceptions by decode. It contains the decoded instruction itself, not the address of the instruction.
0x344	Machine Interrupt Pending (mip)	read/write	bit[11]: meip, machine mode external interrupt pending, read-only. bit[7] mtip, machine mode timer interrupt pending, read-only. bit[3] msip, machine mode software interrupt pending, readable and writable.
0xB00	Machine Cycle (mcycle)	read/write	bit[31:0]: Machine cycle counter
0xB02	Machine Instructions-Retired (minstret)	read/write	bit[31:0]: Machine instructions-retired counter

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

CSR No.	CSR Name	Access	Fields
0xB80	Upper 32 bits of Machine Cycle (mcycleh)	read/write	bit[31:0]: Upper 32 bits of mcycle
0xB82	Upper 32 bits of Machine Instructions-Retired (minstreth)	read/write	bit[31:0]: Upper 32 bits of minstret

2.2.1.9. Composable Extension Unit Logic Interface

CXU-LI defines a set of hardware logic signal interfaces that enable you to connect CPUs and composable extension units (CXU) easily. The term CXU is revised from Custom Function Unit (CFU). In Version 0.91.230803, 2023-08-03 of the RISC-V Composable Custom Extensions Specification, the term Custom Interface (CI) is replaced by Composable Extension (CX). The term CFU is replaced by CXU.

The composable extension unit is a kind of light-weight and customized arithmetic accelerator. With the support of CXU-LI, you can integrate CXUs into your SoC and insert custom functions (CF) to deploy CXU hardware, upon actual solution demand.

In the CXU-LI system, the CPU is the requester and the CXU is the responder. The CPU sends the CXU a request and eventually receives the CXU response. For each request, there is exactly one response.

The CXU-LI is stratified into four separate feature levels:

- L0: combinational;
- L1: fixed latency;
- L2: variable latency;
- L3: reordering.

You can choose an appropriate interface level and design the responder interface of the CXU. For user-friendliness and in compliance with the official spec, the MC core only supports one kind of interface level, L2. It has downward compatibility to support L0 or L1 as well. You can set some signal constant 0 or 1 to degrade L2 to L1 or L0.

The MC core is a -Zicx compatible core, with a mcx_selector CSR added and can repurpose three custom function instruction formats. To deploy the resource of CXU, you only need two steps: interface multiplexing and executing CF instructions.

1. The first step is interface multiplexing, which requires writing a specific selector value to mcx_selector CSR 0xBC0 to select the active CXU and state context.

   The mcx_selector CSR 0xBC0 has the following fields:

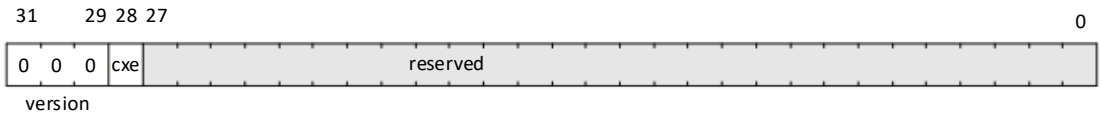

Figure 2.5. mcx_selector CSR 0xBC0 Version 0: Legacy Custom Instructions

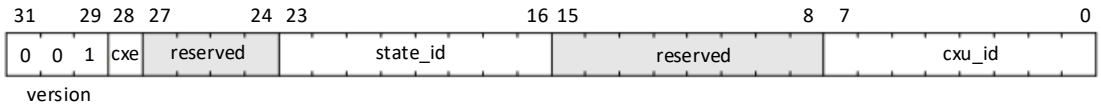

Figure 2.6. mcx_selector CSR 0xBC0 Version 1: Extension Multiplexing

   - version: extension multiplexing version
   - cxe: custom operation exception enable
     - When version=0, disables composable extension multiplexing. When cxe=0, custom-0/1/2/3 instructions execute the CPU's built-in custom instructions and select the CPU's built-in custom CSRs. When cxe=1, custom-0/1/2/3 instruction accesses raise an illegal-instruction exception.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

    - When version=1, enables version-1 composable extension multiplexing. The cxu_id and state_id fields select the current CXU and state context. When cxe=0, custom-0/1/2 instructions issue CXU requests of the CXU and state context identified by cxu_id and state_id. When cxe=1, custom-0/1/2 instruction accesses raise an illegal instruction exception.
    - version values 2-7 are reserved.
  - state_id: selects the hart's current CXU's current state context.
  - cxu_id: selects the hart's current CXU's current state context.

2. The second step is the CPU issuing custom function instructions. When mcx_selector.version=1, the specific function of a CF is defined by customers and identified by custom function identifier, CF_ID. Each CXU packages a set of relevant custom functions. Each CF needs to be implemented by the hardware logic in the CXU. You can design the CXU according to specific scenarios.

   In terms of CF instruction format, three CF formats/major opcodes are reused: custom-0, custom-1, and custom-2. These correspond to three different instructions encoding types: R-type, I-type, and flex-type.

   - Custom-0 R-type encoding
     - Pseudo assembly code: cx_reg cf_id, rd, rs1, rs2
     - An R-type CF instruction issues a CXU request for a zero-extended 10-bit CF_ID cf_id with two source register operands identified by rs1 and rs2. The CXU response data is written to the destination register rd.

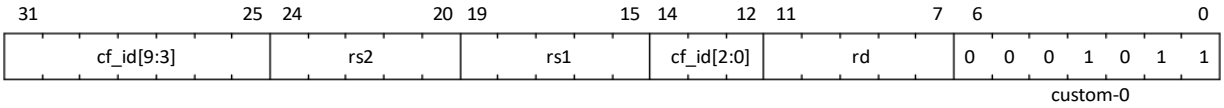

Figure 2.7. CXU R-type Instruction Encoding

   - Custom-1 I-type encoding
     - Pseudo assembly code: cx_imm cf_id, rd, rs1, imm
     - An I-type CF instruction issues a CXU request for a zero-extended 3-bit CF_ID cf_id with one source register operand identified by rs1 and a sign-extended 12-bit immediate value imm. The CXU response is written to the destination register rd.

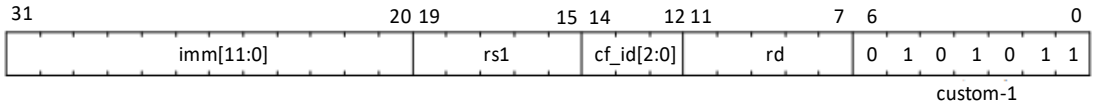

Figure 2.8. CXU I-type Instruction Encoding

   - Custom-2 flex-type encoding
     - Pseudo assembly code: cx_flex cf_id, rs1, rs2
     - Pseudo assembly code: cx_flex25 custom
     - A flex-type CF instruction issues a CXU request for a zero-extended 10-bit CF_ID cf_id with two source register operands identified by rs1 and rs2. There is no destination register and CXU response data is discarded. The instruction is executed purely for its effect upon the selected state context of the selected CXU.

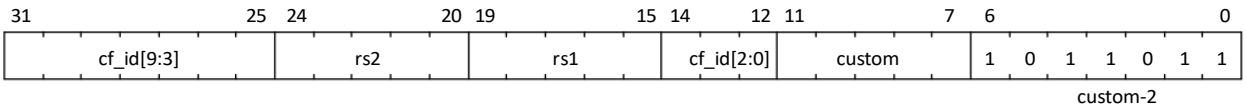

Figure 2.9. CX Flex-type Instruction Encoding

Alternatively, the cx_flex25 form of instruction issues an arbitrary 25-bit custom instruction.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Figure 2.10. CX Flex-type Instruction Alternate Encoding

A flex-type CF instruction may be used with a CXU-L2 request raw instruction field req_insn to provide an arbitrary 25-bit custom request to a CXU. The absence of an integer destination register field is a feature that provides added, CPU-uninterpreted, custom instruction bits to a CXU.

When the CPU issues a custom instruction, it produces a CXU request which has three sources: the fields of instruction, two source operands from the register file and/or an immediate field of instruction, and the cxu_id and state_id fields of mcx_selector (Figure 2.11.). The CXU request may include the CXU_ID, STATE_ID, raw instruction, CF_ID, and operands. The CXU_ID identifies which CXU must process the request. The CXU includes state context(s) and a data path. The STATE_ID selects the state context to use for this request. For custom-0 and custom-1 instructions, the CXU processes the request, possibly updating this state context, and produces a CXU response, which may include the response data. The CPU commits the custom function instructions by writing the response data to the destination register. For custom-2 instructions that do not write response data to the CPU register, the CXU only processes the request, possibly updating this state context. The response data is invalid for the CPU. The CPU commits all the custom-2 instructions by default.

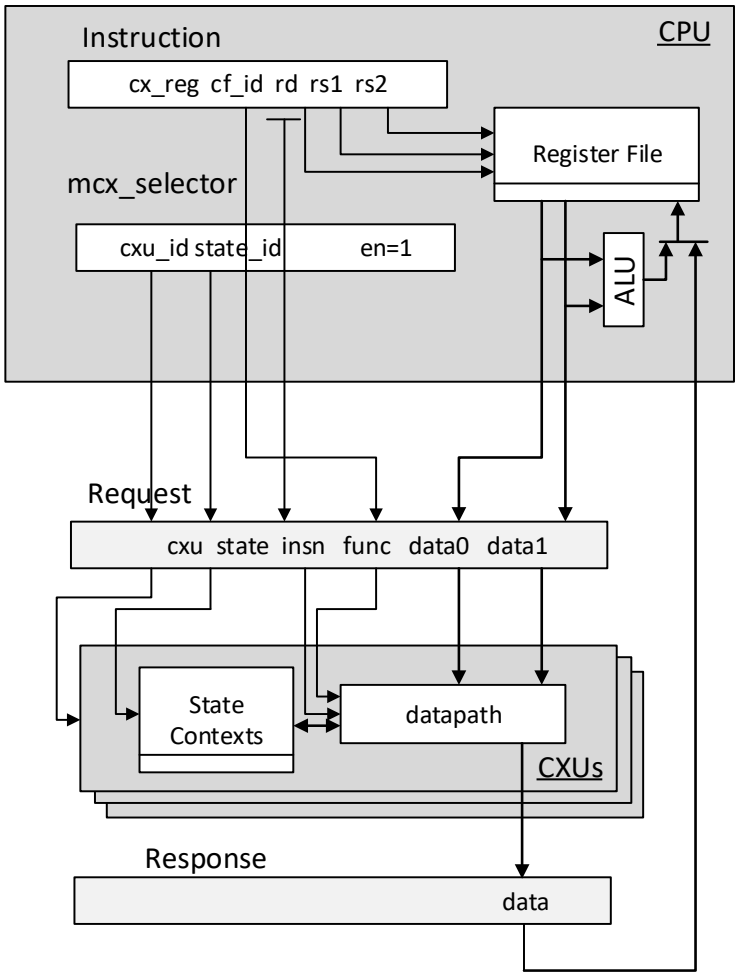

Figure 2.11. Execution of a Custom Function Instruction

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Following is a pseudocode example illustrating CPU issuing stateful CF instructions f0 and f1 to CXU0, f2 and f3 to CXU1, and f4 to CXU0 again.

```
csrw mcx_selector,x20 ; version=1, cxe=0, select CXU_ID=0 and STATE_ID=0
cxu_reg 0,x3,x1,x2 ; u0.f0
cxu_reg 1,x6,x5,x4 ; u0.f1
csrw mcx_selector,x21 ; version=1, cxe=0, select CXU_ID=1 and STATE_ID=0
cxu_reg 2,x9,x7,x8 ; u1.f2
cxu_reg 3,x12,x11,x10 ; u1.f3
csrw mcx_selector,x20 ; version=1, cxe=0, select CXU_ID=0 and STATE_ID=0 again
cxu_reg 4,x15,x13,x14 ; u0.f4
```

1. Write mcx_selector for CXU_ID=0 and STATE_ID=0. Issue two CF instructions to CXU0.
2. Write mcx_selector for CXU_ID=1 and STATE_ID=0. Issue two CF instructions to CXU1.
3. Write mcx_selector for CXU_ID=0 and STATE_ID=0. Issue one CF instruction to CXU0.

2.2.1.10. System Reset Output

The system_resetn_o signal is driven in two ways. When debug is not enabled or if debug reset is not issued, system_resetn_o is the passed value from the input reset signal rst_n_i. It is asynchronous with input clock. When the debugger is enabled and debug reset is issued, the debug reset signal is synchronized to system clock domain and the system_resetn_o is the output of the synchronized signal.

2.2.1.11. RISC-V Formal Interface

The RISC-V Formal Interface is supported. This interface can help you get many important information directly, including the privilege mode, trap, instruction, and so on.

2.2.2. Submodule

The CPU soft IP contains two submodules: PIC and Timer. The PIC and Timer share the same start address in the memory map and a fixed 2 KB address range is allocated, if either the PIC or Timer attribute is enabled under the Interrupt tab in the Module/IP Block Wizard GUI (Figure 2.12).

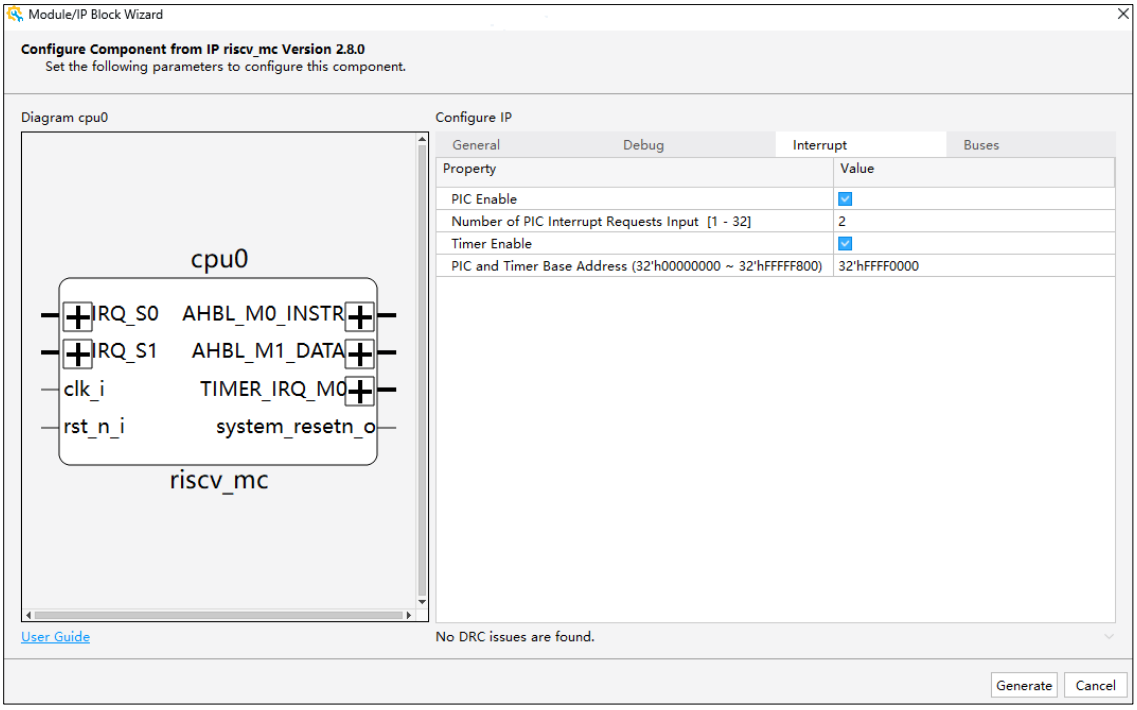

Figure 2.12. Interrupt Tab

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.2.2.1. PIC

The PIC aggregates up to eight external interrupt inputs (IRQs) into one interrupt output to the processor core. The interrupt status register can be used to read the values of IRQs. Individual IRQs can be configured by programming the corresponding PIC_STATUS, PIC_ENABLE, PIC_SET, and PIC_POL registers. All registers can be accessed through the CPU's internal AHB-Lite interface, as shown in Figure 2.13.

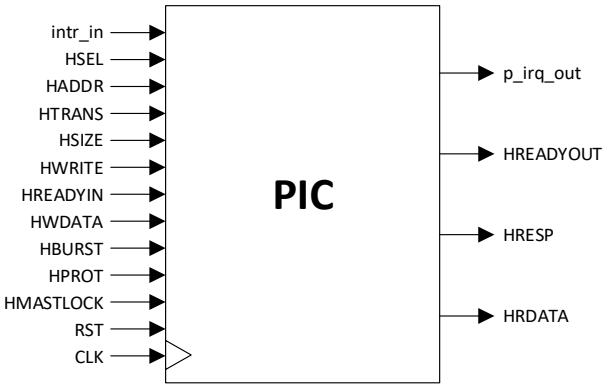

Figure 2.13. PIC Block Diagram

Table 2.2 provides the description of PIC registers.

Table 2.2. PIC Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x000</td><td>PIC_STATUS</td><td>Interrupt Status Register
Access: read-write
Parameterizable width: min=2, max=32
Indicates the pending interrupt at corresponding interrupt request port (irq[i] at top level).

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_STATUS [N-1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_STATUS [1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_STATUS [0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>

PIC_STATUS[i]:
Read
• 0 – No interrupt at irq[i]
• 1 – Interrupt pending at irq[i]
Write
• 0 – No effect
• 1 – Clear interrupt status for irq[i]</td></tr>
<tr><td>0x004</td><td>PIC_ENABLE</td><td>Interrupt Enable Register
Access: read-write
Parameterizable width: min=2, max=32
Indicates whether the processor responds to the interrupt from corresponding interrupt request port (irq[i]) or not.

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_ENABLE[N-1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_ENABLE[1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_ENABLE[0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>
</td></tr>
</table>

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td></td><td></td><td>PIC_ENABLE[i]:
Read
• 0 – irq[i] disabled
• 1 – irq[i] enabled
Write
• 0 – Disable irq[i]
• 1 – Enable irq[i]</td></tr>
<tr><td>0x008</td><td>PIC_SET</td><td>Interrupt Set Register
Access: write-only
Parameterizable width: min=2, max=32
Sets the interrupt status for corresponding interrupt request port (irq[i]).

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N-1]</td><td>PIC_SET [N-1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_SET [1]</td><td>W</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_SET [0]</td><td>W</td><td>1</td><td>0x0</td></tr>
</table>

PIC_SET[i]:
Read
• Invalid operation gets 0.
Write
• 0 – No effect
• 1 – Set interrupt status for irq[i] (set PIC_STATUS[i]).</td></tr>
<tr><td>0x00C</td><td>PIC_POL</td><td>Interrupt Polarity Register
Access: read-write
Parameterizable width: min=2, max=32
Indicates the polarity of corresponding interrupt request (irq[i]) port.

<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[N]</td><td>PIC_POL [N]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td>[1]</td><td>PIC_POL I[1]</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>PIC_POL [0]</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>

PIC_POL[i]:
Read
• 0 – irq[i] active high
• 1 – irq[i] active low
Write
• 0 – Set irq[i] active high
• 1 – Set irq[i] active low</td></tr>
</table>

Note: The register definition of PIC follows Lattice Interrupt Interface (LINTR) Standard, refer to Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039) for more information.

2.2.2.2. Timer

The Timer module provides a 64-bit real-time counter register, mtime, and time compare register, mtimecmp. An output interrupt signal notifies the RISC-V processor core when the value of mtime is greater than or equal to the value of mtimecmp. All registers can be accessed through the CPU's internal AHB-Lite interface, as shown in Figure 2.14.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

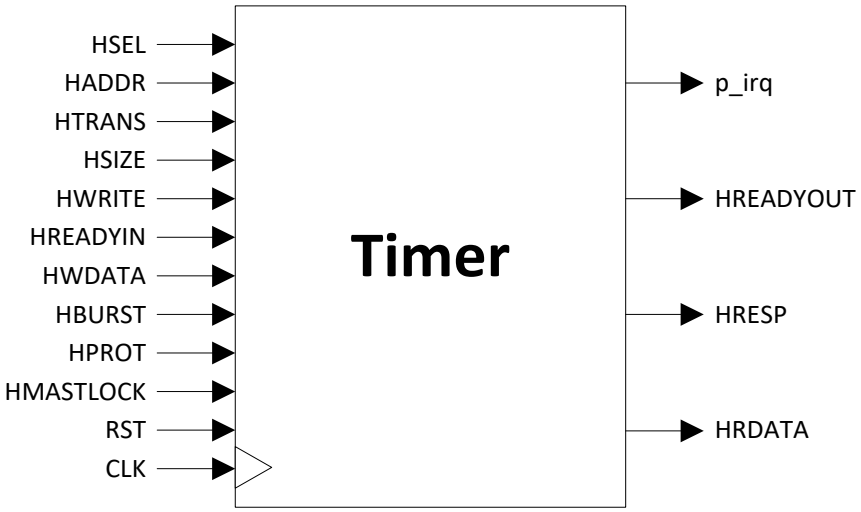

Figure 2.14. Timer Block Diagram

Table 2.3 provides the description of Timer registers.

Table 2.3. Timer Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x400</td><td>TIMER_CNT_L</td><td>Lower 32 bits of Timer counter register.

<table><tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr><tr><td>[63:0]</td><td>mtime</td><td>RW</td><td>64</td><td>0x0</td></tr></table>
mtime
A 64-bit real-time counter register. You must set the register to a non-zero value to start the counting process.</td></tr>
<tr><td>0x404</td><td>TIMER_CNT_H</td><td>Higher 32 bits of Timer counter register.</td></tr>
<tr><td>0x410</td><td>TIMER_CMP_L</td><td>Lower 32 bits for Timer time compare register.

<table><tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr><tr><td>[63:0]</td><td>mtimecmp</td><td>RW</td><td>64</td><td>0x0</td></tr></table>
mtimecmp
This register is used to generate or clear the timer interrupt, mtip. When the value of mtime register is greater than or equal to the value of mtimecmp register, TIMER_IRQ_M0 is asserted.
The interrupt remains posted until mtimecmp becomes greater than mtime, typically as a result of writing mtimecmp.</td></tr>
<tr><td>0x414</td><td>TIMER_CMP_H</td><td>Higher 32 bits for Timer time compare register.</td></tr>
</table>

2.3. Signal Description

Table 2.4 to Table 2.7 list the ports of the CPU soft IP in different categories. As shown in Figure 2.15, the AHB-Lite, CXU-LI, and RVFI interfaces can be configured and enabled under the Buses tab in the Module/IP Block Wizard. The Interrupt Interface can be configured under the Interrupt tab, as shown in Figure 2.12.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

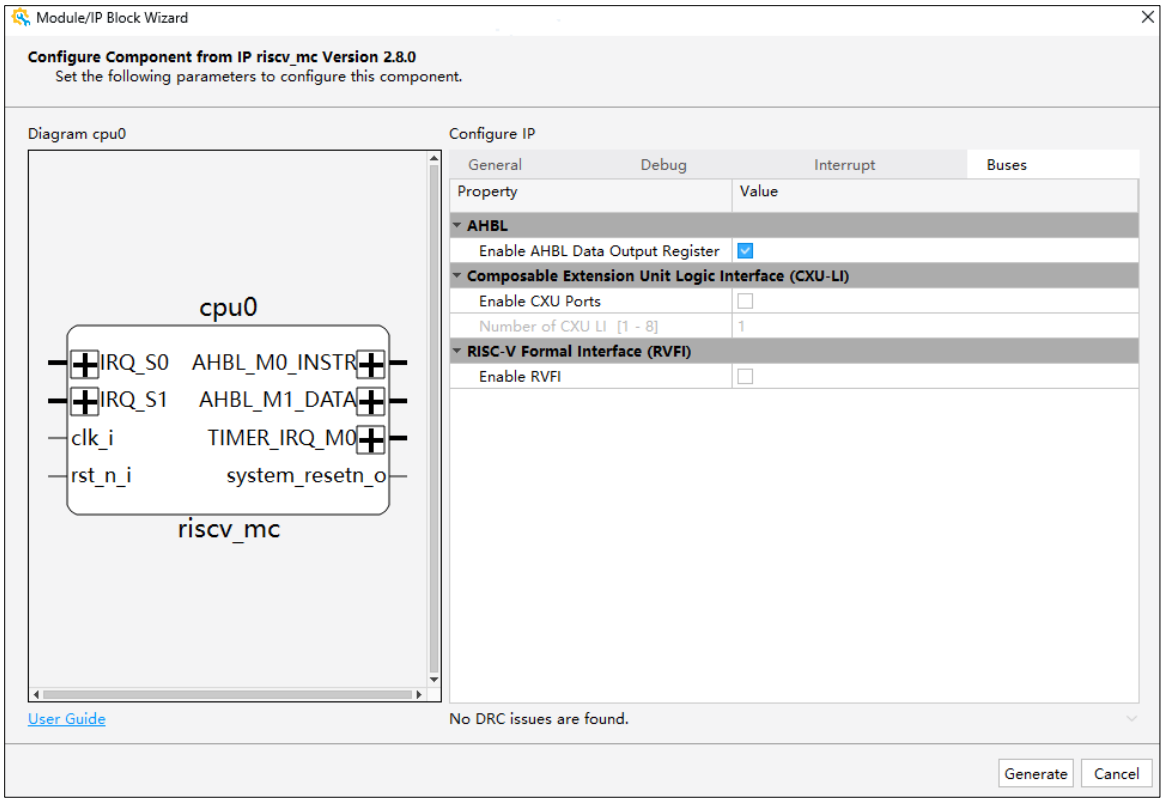

Figure 2.15. Buses Tab

2.3.1. Clock and Reset

Table 2.4. Clock and Reset Ports

Name	Direction	Width	Description
clk_i	In	1	RISC-V soft IP clock
rst_n_i	In	1	Global reset, active low
system_resetn_o	Out	1	Combined Global reset and Debug Reset from JTAG, active low

2.3.2. Instruction and Data Interface

Table 2.5. Instruction Ports

Name	Direction	Width	Description
AHBL_M0_INSTR - HADDR	Out	32	—
AHBL_M0_INSTR - HWRITE	Out	1	Fixed at 1’b0
AHBL_M0_INSTR - HSIZE	Out	3	Fixed at 3’b010
AHBL_M0_INSTR - HPROT	Out	4	Fixed at 4’b1110 when caches are not enabled.
AHBL_M0_INSTR - HTRANS	Out	2	—
AHBL_M0_INSTR - HBURST	Out	3	—
AHBL_M0_INSTR - HMASTLOCK	Out	1	Fixed at 1’b0
AHBL_M0_INSTR - HWDATA	Out	32	—
AHBL_M0_INSTR - HRDATA	In	32	—
AHBL_M0_INSTR - HREADY	In	1	—
AHBL_M0_INSTR - HRESP	In	1	—

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.6. Data Ports

Name	Direction	Width	Description
AHBL_M1_DATA - HADDR	Out	32	—
AHBL_M1_DATA - HWRITE	Out	1	—
AHBL_M1_DATA - HSIZE	Out	3	—
AHBL_M1_DATA - HPROT	Out	4	Fixed at 4'b1111 when caches are not enabled.
AHBL_M1_DATA - HTRANS	Out	2	—
AHBL_M1_DATA - HBURST	Out	3	—
AHBL_M1_DATA - HMASTLOCK	Out	1	—
AHBL_M1_DATA - HSEL	Out	1	—
AHBL_M1_DATA - HWDATA	Out	32	—
AHBL_M1_DATA - HRDATA	In	32	—
AHBL_M1_DATA - HREADY	In	1	—
AHBL_M1_DATA - HRESP	In	1	—

Note: Refer to AMBA 3 AHB-Lite Protocol V1.0 for more information.

2.3.3. Interrupt Interface

Table 2.7. Interrupt Ports

Name	Type	Width	Description
IRQ_S_{0-31}	In	1	Peripheral interrupts.
TIMER_IRQ_M0	Out	1	Timer interrupt output, active high. It exists only when TIMER_ENABLE is checked.
TIMER_IRQ_S0	In	1	Timer interrupt input, active high. It exists only when TIMER_ENABLE is unchecked.

2.3.4. CXU-LI Interface

CXU-LI is used to connect a CXU accelerator. The RISC-V core supports up to eight CXU-LIs. You can enable CXU-LI and configure the number of CXU-LI in the Module/IP Block Wizard GUI.

Table 2.8. CXU-LI Ports, Optional

Port	Direction	Width	Group	Description
req_valid	out	1	Request	Request valid
req_ready	in	1		Request ready
req_cxu	out	4		Request CXU_ID
req_state	out	3		Request STATE_ID
req_func	out	3		Request CF_ID
req_insn	out	32		Request raw instruction
req_data0	out	32		Request operand data 0
req_data1	out	32		Request operand data 1
resp_valid	in	1	Response	Response valid
resp_ready	out	1		Response ready
resp_status	in	3		Response status
resp_data	in	32		Response data

2.3.5. RVFI Interface

The MC core supports the instruction metadata, integer register read/write, program counter, and memory access signals of the RISC-V Formal Interface.

The Interface consists of only output signals. As shown in Figure 2.16, when the core retires an instruction, it asserts the rvfi_valid signal and uses the signals described in Table 2.9 to output the details of the retired instruction. The

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

signals below are only valid during such a cycle and can be driven to arbitrary values in a cycle in which rvfi_valid is not asserted.

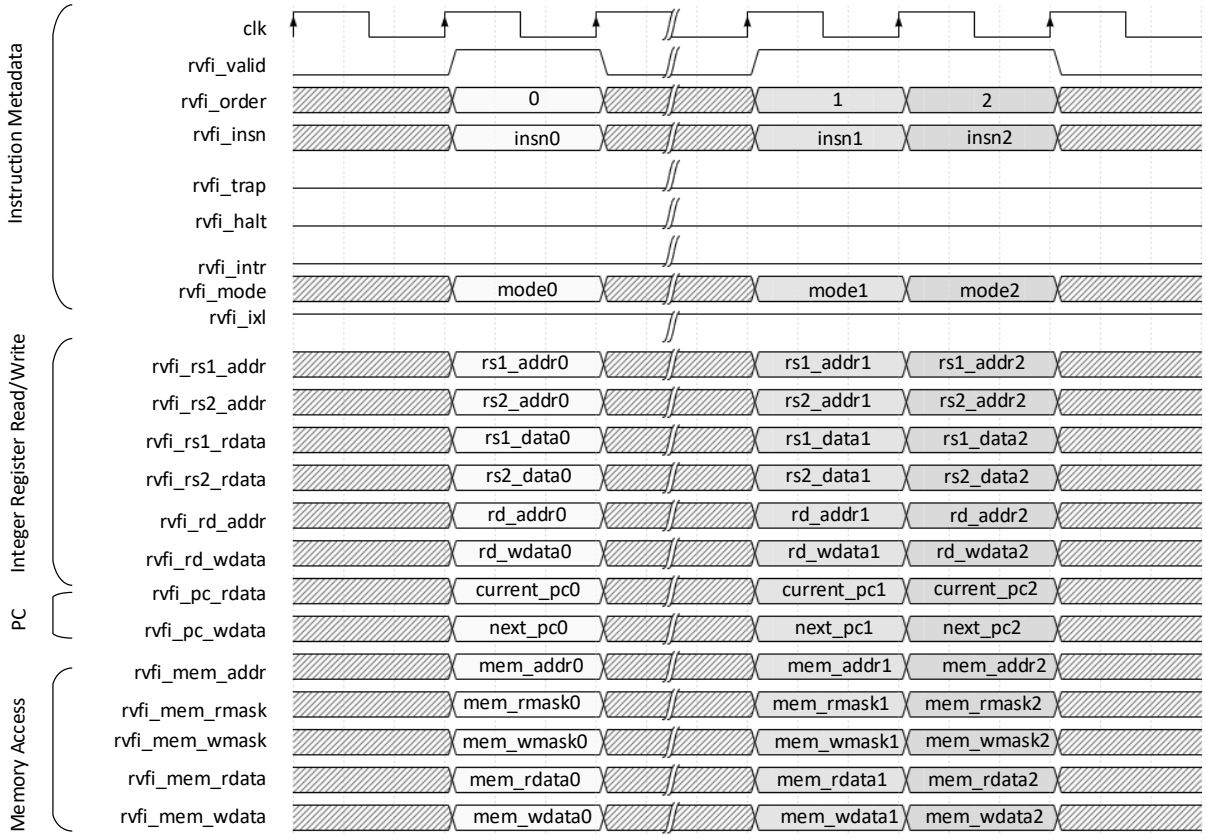

Figure 2.16. RVFI Interface

Table 2.9. RVFI Ports, Optional

Port	Direction	Width	Group	Description
rvfi_valid	out	1	Instruction Metadata	The core retires an instruction and the following signals also become valid if rvfi_valid is valid.
rvfi_order	out	64		Instruction index.
rvfi_insn	out	32		Instruction word for the retired instruction.
rvfi_trap	out	1		Fixed at 1'b0.
rvfi_halt	out	1		Fixed at 1'b0.
rvfi_intr	out	1		Fixed at 1'b0.
rvfi_mode	out	2		Current privilege level.
rvfi_ixl	out	2		The value of MXL/SXL/UXL fields in the current privilege level.
rvfi_rs1_addr	out	5	Integer Register Read/Write	The decoded rs1 register addresses for the retired instruction.
rvfi_rs2_addr	out	5		The decoded rs2 register addresses for the retired instruction.
rvfi_rs1_rdata	out	32		The value of the x register addressed by rs1 before the execution of this instruction.
rvfi_rs2_rdata	out	32		The value of the x register addressed by rs2 before the execution of this instruction.
rvfi_rd_addr	out	5		The decoded rd register address for the retired instruction.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Port	Direction	Width	Group	Description
rvfi_rd_wdata	out	32		The value of the x register addressed by rd after execution of this instruction.
rvfi_pc_rdata	out	32	Program Counter	The address of the retired instruction.
rvfi_pc_wdata	out	32		The address of the next instruction.
rvfi_mem_addr	out	32	Memory Access	Holds the accessed memory location, the address is 4-byte alignment.
rvfi_mem_rmask	out	4		A bitmask that specifies which bytes in rvfi_mem_rdata contain valid read data from rvfi_mem_addr.
rvfi_mem_wmask	out	4		A bitmask that specifies which bytes in rvfi_mem_wdata contain valid data that is written to rvfi_mem_addr.
rvfi_mem_rdata	out	32		The pre-state data read from rvfi_mem_addr.
rvfi_mem_wdata	out	32		The post-state data written to rvfi_mem_addr.

2.4. Attribute Summary

The configurable attributes of the RISC-V MC CPU IP are shown in Table 2.10. and are described in Table 2.11.

The attributes can be configured through the Lattice Propel Builder software.

Table 2.10. Configurable Attributes

Attribute	Selectable Values	Default	Dependency on Other Attributes
Extension			
E Extension for Embedded Device	Enabled, Disabled	Disabled	—
C Extension for Compressed Instructions	Enabled, Disabled	Enabled	Selectable when E Extension is disabled.
	Disabled	Disabled	When E Extension is enabled, C Extension is fixed as disabled.
M Extension for Integer Mult and Div	Enabled, Disabled	Disabled	Selectable when C Extension is enabled and E Extension disabled. For Certus-N2, Lattice Avant, Certus-NX, CertusPro-NX, CrossLink-NX, and MachXO5-NX devices only.
	Disabled	Disabled	When E Extension is enabled, M Extension is fixed as disabled.
Reset			
Reset Vector	32'h00000000—32'hFFFFFFFF	32'h00000000	—
Cache			
Cache Enable	Enabled, Disabled	Disabled	—
Instruction Cache Enable	Enabled, Disabled	Disabled	Enabled when Cache Enable is enabled.
Data Cache Enable	Enabled, Disabled	Disabled	Enabled when Cache Enable is enabled.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Attribute	Selectable Values	Default	Dependency on Other Attributes
Instruction Cache Cacheable Address Range Low Limit	32'h00000000–32'hFFFFFFFF	32'h00000000	Editable when Cache Enable is enabled.
Instruction Cache Cacheable Address Range High Limit	32'h00000000–32'hFFFFFFFF	32'hF0000000	Editable when Cache Enable is enabled.
Data Cache Cacheable Address Range Low Limit	32'h00000000–32'hFFFFFFFF	32'h00000000	Editable when Cache Enable is enabled.
Data Cache Cacheable Address Range High Limit	32'h00000000–32'hFFFFFFFF	32'hF0000000	Editable when Cache Enable is enabled.
Debug			
Debug Enable	Enabled, Disabled	Enabled	—
Soft JTAG	Enabled, Disabled	Disabled	—
Extend JTAG Channel	Enabled, Disabled	Disabled	—
JTAG Channel Selection for Certain Devices	14–24	14	Editable when Debug Enable is checked. When Extend JTAG Channel is enabled, for Certus-N2 and Lattice Avant devices, the channel range enlarges from 14–16 to 14–24. For nexus family devices, the channel range enlarges from 14–16 to 10–18.
	10–18	14	
Interrupt			
PIC Enable	Enabled, Disabled	Enabled	—
Number of PIC Interrupt Requests Input	0–32	8	The value is configurable from 1–32 when PIC Enable is enabled. The value is fixed at 0 when PIC Enable is disabled.
Timer Enable	Enabled, Disabled	Enabled	—
PIC and Timer Base Address	32'h00000000–32'hFFFFF800	32’hFFFF0000	Editable when PIC Enable, or Timer Enable is enabled.
AHBL			
Enable AHBL Data Output Register	Enabled, Disabled	Disabled	Editable when Cache Enable is disabled.
CXU-LI			
Enable CXU Ports	Enabled, Disabled	Disabled	—
Number of CXU-LI	1–8	1	Editable when Enable CXU Ports is enabled.
RVFI			
Enable RVFI	Enabled, Disabled	Disabled	—

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.11. Attributes Description

Attribute	Description
Extension	
E Extension for Embedded Device	Enabled: Supports E Extension. Disabled: Does not support E Extension.
C Extension for Compressed Instructions	Enabled: Supports compressed instructions. Disabled: Does not support compressed instructions.
M Extension for Integer Mult and Div	Enabled: Supports M standard extension. Disabled: Does not support M standard extension.
Reset	
Reset Vector	The value of reset vector.
Cache	
Cache Enable	Enabled: The Cache function is enabled. Disabled: The Cache function is disabled.
Instruction Cache Enable	Enabled: The Instruction cache is enabled. Disabled: The Instruction cache is disabled.
Data Cache Enable	Enabled: The Data cache is enabled. Disabled: The Data cache is disabled.
Instruction Cache Cacheable Address Range Low Limit	Lower limit of the cacheable address range for the instruction cache. This address itself is included.
Instruction Cache Cacheable Address Range High Limit	Higher limit of the cacheable address range for the instruction cache. This address itself is included.
Data Cache Cacheable Address Range Low Limit	Lower limit of the cacheable address range for the data cache. This address itself is included.
Data Cache Cacheable Address Range High Limit	Higher limit of the cacheable address range for data cache. This address itself is included.
Debug	
Debug Enable	Enabled: The debug function is enabled. Disabled: The debug function disabled.
Soft JTAG	Enabled: Debug with hard JTAG. Disabled: Debug with soft JTAG.
Extend JTAG Channel	Enables the JTAG channel's range extension.
JTAG Channel Selection for Certain Devices	Specifies the channel of MC JTAG block.
Interrupt	
PIC Enable	Enabled: PIC is enabled. Disabled: PIC is disabled.
Number of PIC Interrupt Requests Input	Number of peripheral interrupts
Timer Enable	Enabled: Timer is enabled. Disabled: Timer is disabled.
PIC and Timer Base Address	Start address of the PIC and Timer
AHBL	
Enable AHBL Data Output Register	Enabled: AHB-Lite data output register is enabled. Disabled: AHB-Lite data output register is disabled.
CXU-LI	
Enable CXU Ports	Enables CXU. When CXU is enabled, the Debug Enable attribute is fixed as enabled.
Number of CXU-LI	CXU interface number

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Attribute	Description
RVFI	
Enable RVFI	Enables RVFI.

Notes:

- Instruction Cache Enable and Data Cache Enable should only be enabled when using Certus-N2, Lattice Avant, MachXO5-NX, CrossLink-NX, CertusPro-NX, and Certus-NX devices with sufficient resources. For MachXO2, MachXO3L, MachXO3LF, and MachXO3D devices, cache features are not supported as their resources are limited.
- Instruction Cache Cacheable Address Range Lower Limit should not be larger than Instruction Cache Cacheable Address Range Higher Limit, and address range between Instruction Cache Cacheable Address Range Low Limit and Instruction Cache Cacheable Address Range High Limit should not be overlapped with peripheral device address ranges. The memory which stores instructions should always be fixed in this range.
- Similarly, Data Cache Cacheable Address Range Low Limit should not be larger than Data Cache Cacheable Address Range High Limit, and address range between Data Cache Cacheable Address Range Low Limit and Data Cache Cacheable Address Range High Limit should not be overlapped with peripheral devices address ranges.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. RISC-V MC CPU IP Generation

This section provides information on how to generate the MC CPU IP using Lattice Propel Builder.

To generate the MC CPU IP:

1. In Lattice Propel Builder, create a new design. Select the CPU package. Enter the component name, as shown in Figure 3.1. Click **Next**.

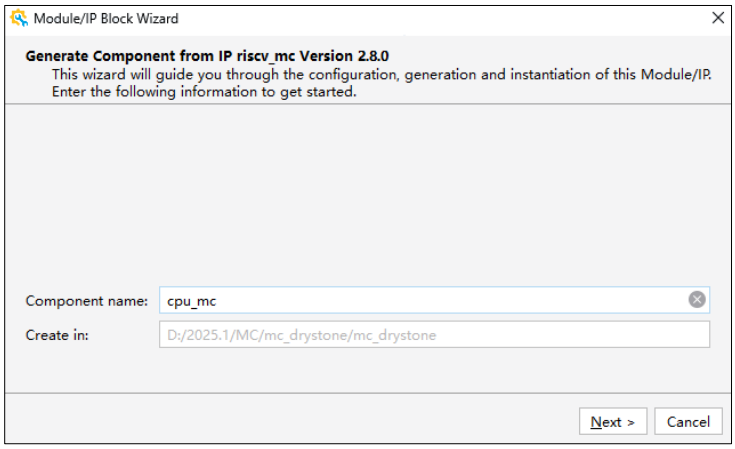

Figure 3.1. Entering Component Name

2. Configure the parameters, as shown in Figure 3.2. Click **Generate**.

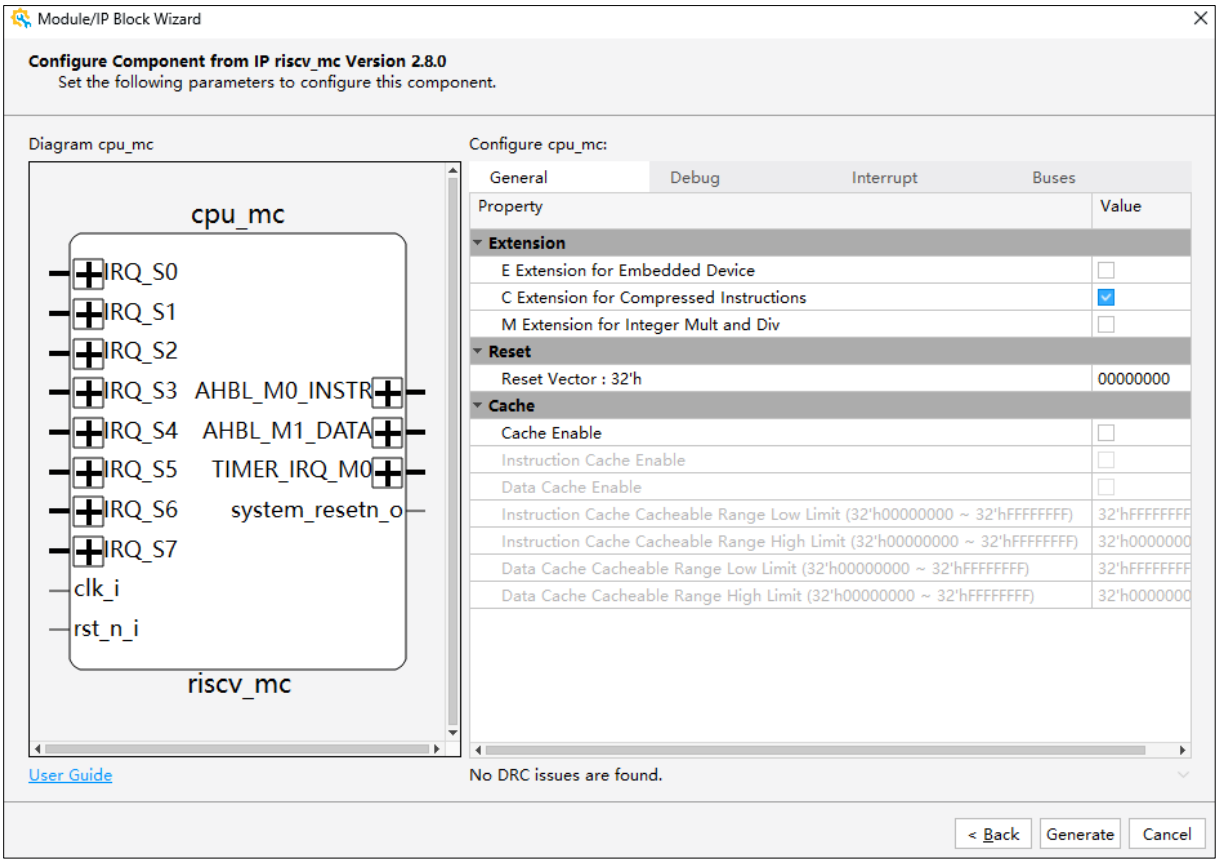

Figure 3.2. Configuring Parameters

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Verify the information. Click **Finish**.

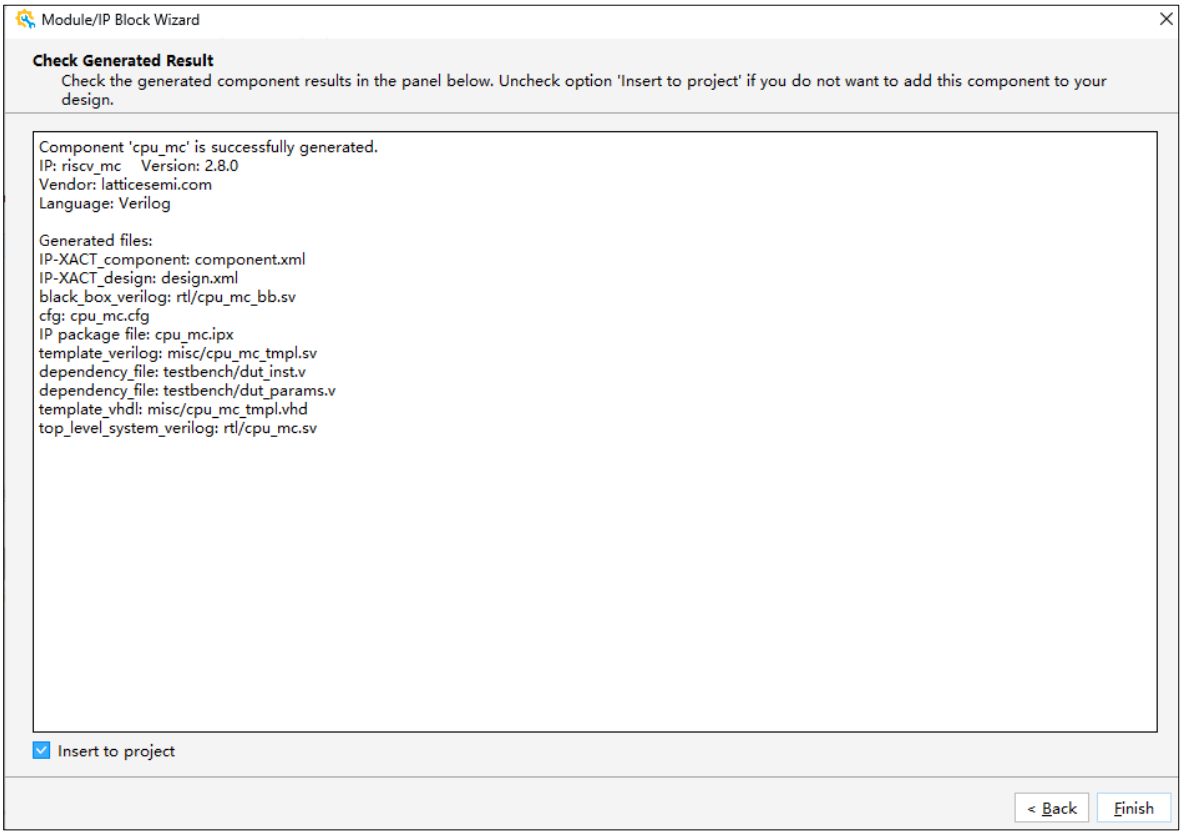

Figure 3.3. Verifying Results

4. Confirm or modify the module instance name. Click **OK**.

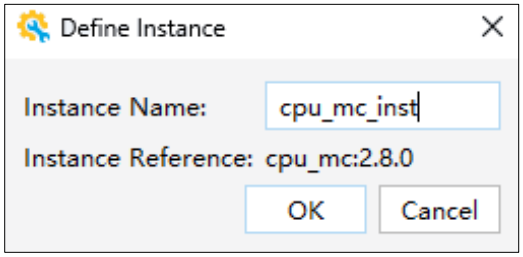

Figure 3.4. Specifying Instance Name

The CPU IP instance is successfully generated, as shown in Figure 3.5.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

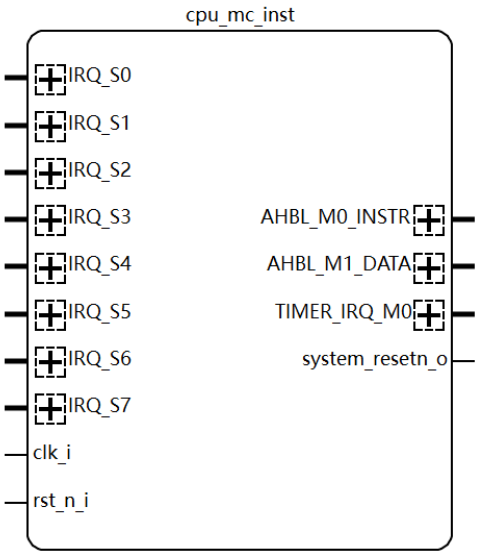

Figure 3.5. Generated Instance

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix A. Resource Utilization

Table A.1. Resource Utilization in MachXO3D Device, with Cache Disabled

Configuration	LUTs	Registers	EBRs
Processor core only	1450	806	4
Processor core + PIC	1559	845	4
Processor core + Timer	1844	955	4
Processor core + Debug	1726	1108	4
Processor core + C_EXT	1738	860	4
Processor core + PIC + Timer	1927	989	4
Processor core + PIC + Timer + Debug	2162	1287	4
Processor core + PIC + Timer + Debug + C_EXT	2385	1345	4

Note: Resource utilization characteristics are generated using Lattice Diamond software.

Table A.2. Resource Utilization in CrossLink-NX Device, with Cache Disabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core only	1620	821	2	0
Processor core + PIC	1789	859	2	0
Processor core + Timer	2103	959	2	0
Processor core + Debug	1979	1194	2	0
Processor core + C_EXT	1911	830	2	0
Processor core + C_EXT + M_EXT	2417	1172	2	6
Processor core + PIC + Timer	2226	994	2	0
Processor core + PIC + Timer + Debug	2494	1375	2	0
Processor core + PIC + Timer + Debug + C_EXT	2790	1450	2	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

Table A.3. Resource Utilization in CrossLink-NX Device, with Cache Enabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core + only	3372	1423	16	0
Processor core + PIC	3568	1479	16	0
Processor core + Timer	3753	1574	16	0
Processor core + Debug	3687	1810	16	0
Processor core + C_EXT	3711	1551	16	0
Processor core + C_EXT + M_EXT	4133	1879	16	6
Processor core + PIC + Timer	3894	1606	16	0
Processor core + PIC + Timer + Debug	4309	1980	16	0
Processor core + PIC + Timer + Debug + C_EXT	4568	2032	16	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table A.4. Resource Utilization in Lattice Avant Device, with Cache Disabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core + only	2015	867	2	0
Processor core + PIC	2097	886	2	0
Processor core + Timer	2456	979	2	0
Processor core + Debug	2376	1268	2	0
Processor core + C_EXT	2202	860	2	0
Processor core + C_EXT + M_EXT	2731	1284	2	6
Processor core + PIC + Timer	2535	1017	2	0
Processor core + PIC + Timer + Debug	2869	1444	2	0
Processor core + PIC + Timer + Debug + C_EXT	3128	1528	2	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

Table A.5. Resource Utilization in Lattice Avant Device, with Cache Enabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core + only	3340	1364	18	0
Processor core + PIC	3437	1406	18	0
Processor core + Timer	3857	1548	18	0
Processor core + Debug	3708	1806	18	0
Processor core + C_EXT	3671	1546	18	0
Processor core + C_EXT + M_EXT	4431	1861	18	6
Processor core + PIC + Timer	3809	1655	18	0
Processor core + PIC + Timer + Debug	4381	1995	18	0
Processor core + PIC + Timer + Debug + C_EXT	4559	2168	18	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

Table A.6. Resource Utilization in CertusPro-NX Device, with Cache Disabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core + only	1978	834	2	0
Processor core + PIC	2105	886	2	0
Processor core + Timer	2446	990	2	0
Processor core + Debug	2304	1218	2	0
Processor core + C_EXT	2288	896	2	0
Processor core + C_EXT + M_EXT	2790	1185	2	6
Processor core + PIC + Timer	2578	1032	2	0
Processor core + PIC + Timer + Debug	2871	1390	2	0
Processor core + PIC + Timer + Debug + C_EXT	3220	1461	2	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

Table A.7. Resource Utilization in CertusPro-NX Device, with Cache Enabled

Configuration	LUTs	Registers	EBRs	DSP
Processor core + only	3031	1461	20	0
Processor core + PIC	3391	1505	20	0
Processor core + Timer	3812	1607	20	0
Processor core + Debug	3667	1854	20	0
Processor core + C_EXT	3691	1613	19	0
Processor core + C_EXT + M_EXT	4334	2007	19	6
Processor core + PIC + Timer	3926	1671	20	0
Processor core + PIC + Timer + Debug	4139	2012	20	0
Processor core + PIC + Timer + Debug + C_EXT	4450	2169	19	0

Note: Resource utilization characteristics are generated using Lattice Radiant software.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Appendix B. Debug with Soft JTAG

To debug with Soft JTAG:

1. In Lattice Propel Builder, enable **Soft JTAG** in the Module/IP Block Wizard GUI (Figure 3.2) when generating the IP.
2. After the IP is generated, right-click on the **JTAG** port and select **Export** (Figure B.1).

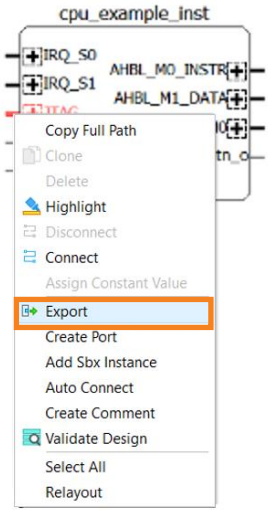

Figure B.1. Exporting Pins

3. Assign the normal I/O as JTAG I/O using the Device Constraint Editor in Lattice Radiant software.
   a. Synthesize the design SoC in Lattice Radiant software by clicking **Synthesis Design** from the process toolbar.
   b. Open **Device Constraint Editor** from the **Tools** tab in Lattice Radiant software and assign the pins. For different devices, refer to the user guide of each board. The following assignment is for LFCPNX-100-9LF672C (Figure B.2).

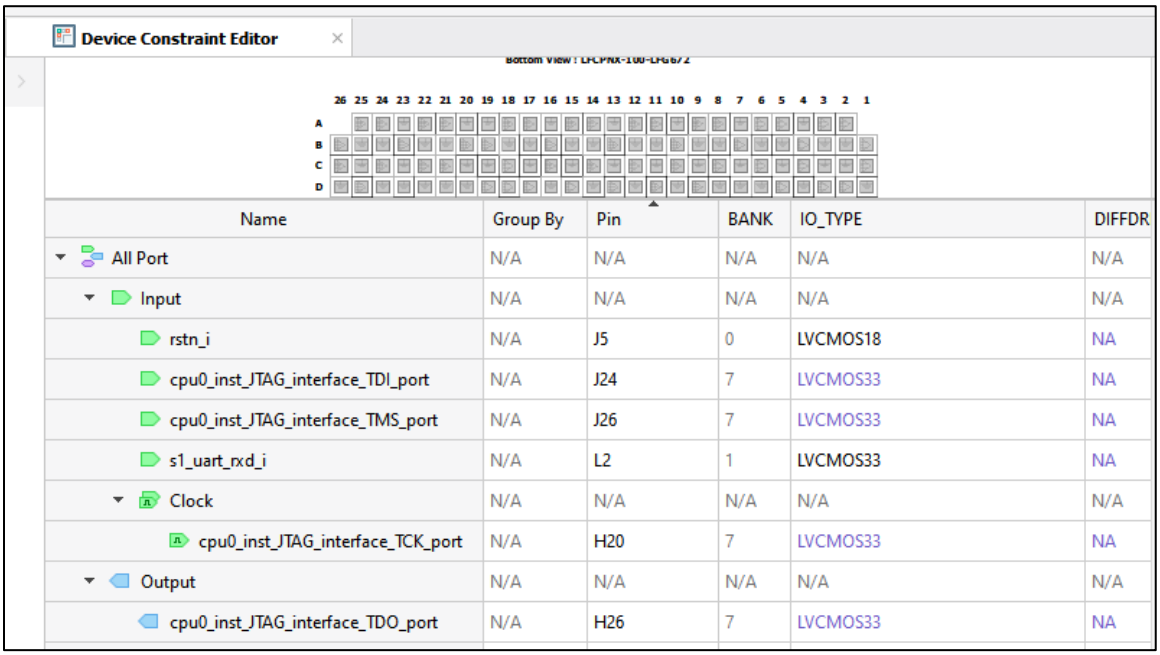

Figure B.2. Assigning Pins

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

c. Double-click on the targeted strategy in the **File List** view to open the **Strategies** dialog box.

d. In the **Strategies** dialog box, set the environment variable for **Place & Route Design**. Enter "-exp WARNING_ON_PCLKPLC1=1" in the Value of **Command Line Options** if TCK connects to normal I/O (Figure B.3).

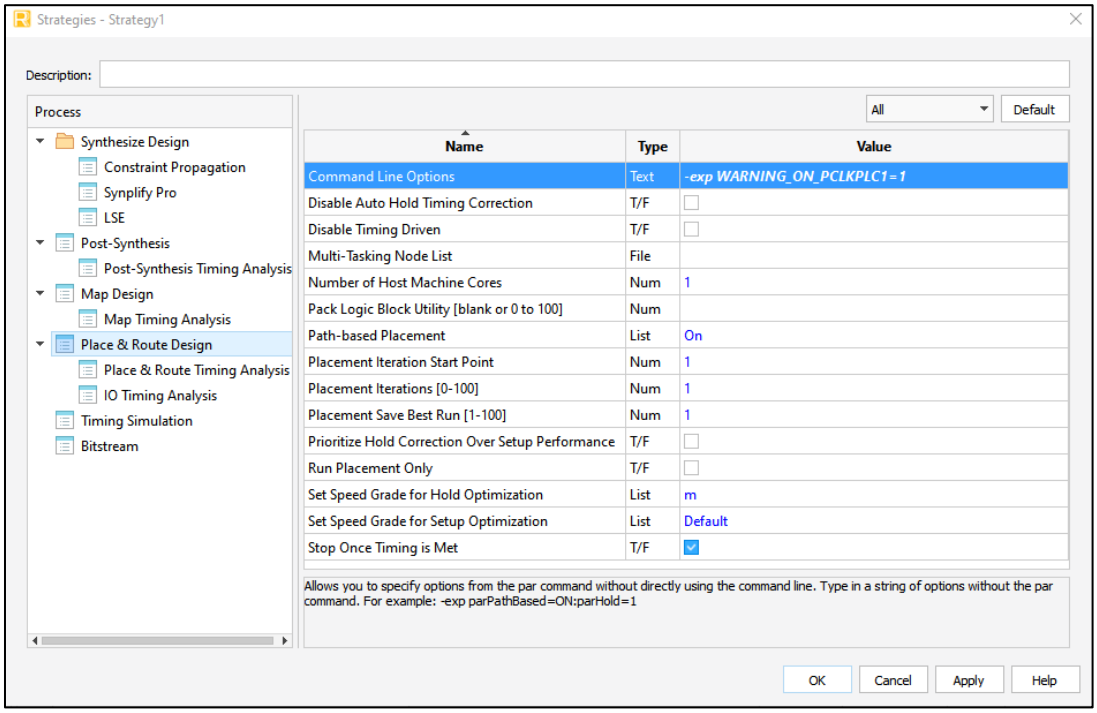

Figure B.3. Setting Environment Variables

e. Generate the bitstream and load it to the board.

f. Connect the pins on cable to the board according to your assignments. Connect VCC and GND. Scan the cable in Lattice Propel SDK software and ignore the scanning of the device.
Note: C projects generated for Certus-N2 and Lattice Avant family devices cannot use Soft JTAG to debug on MachXO5-NX, Certus-NX, CertusPro-NX, and CrossLink-NX boards and vice versa.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

References

- Lattice Propel 2025.1 Builder User Guide (FPGA-UG-02235)
- AMBA 3 AHB-Lite Protocol V1.0
- RISC-V Privileged Specification (20211203)
- RISC-V Instruction Set Manual (20190608)
- RISC-V Composable Custom Extensions Specification (Draft)
- Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039)

For more information, refer to:

- Lattice Propel web page
- Certus-N2 Family Devices web page
- Lattice Avant-E Family Devices web page
- Lattice Avant-G Family Devices web page
- Lattice Avant-X Family Devices web page
- MachXO5-NX Family Devices web page
- Certus-NX Family Devices web page
- CertusPro-NX Family Devices web page
- CrossLink-NX Family Devices web page
- MachXO3D Family Devices web page
- MachXO3 Family Devices web page
- MachXO2 Family Devices web page
- Lattice Insights for Lattice Semiconductor Training Series and Learning Plans

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Revision History

Revision 1.1, IP v2.8.0, July 2025

Section	Change Summary
Functional Description	• In the Exception section, changed *A write error on the local and AXI bus on the processor causes the Store/Atomic Memory Operation (AMO) access fault exception of the core, with exception ID 7* to *A write error on the AHB-L data bus causes the store access exception with ID 7*. • In Table 2.1. RISC-V Processor Core Control and Status Registers, added *7 – write access fault* to Fields of the Machine Cause (mcause) control and status register.

Revision 1.0, IP v2.8.0, June 2025

Section	Change Summary
All	Production release.

© 2025 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

www.latticesemi.com