RISC-V RX CPU IP – Lattice Propel Builder 2024.2

IP Version: v2.5.0

User Guide

FPGA-IPUG-02269-1.0

November 2024

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.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
ABI	Application Binary Interface
AEE	Application Execution Environment
AMO	Atomic Memory Operation
AXI	Advanced Extensible Interface
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
CLINT	Core Local Interrupter
CPU	Central Processing Unit
CSR	Control and Status Register
DDR	Double Data Rate
DMIPS	Dhrystone Million Instructions per Second
DTCM	Data TCM
EIP	External Interrupt Pending
FPGA	Field Programmable Gate Array
GDB	Gnu Debugger
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDL	Hardware Description Language
IE	Interrupt Enable
IP	Intellectual Property
IRQ	Interrupt Request
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
ITCM	Instruction TCM
LUT	Lookup-Table
misa	Machine Instruction Set Architecture Register
NMI	Non-Maskable Interrupt
OpenOCD	Open On-Chip Debugger
OS	Operating System
OSC	Oscillator
PC	Personal Computer
PLIC	Platform-Level Interrupt Controller
PLL	Phase-Locked Loop
PMP	Physical Memory Protection
RISC-V	Reduced Instruction Set Computer-V (five)
RV32IMC	RISC-V Integer, M & Compressed Instruction Sets
RX	Real Time OS, RISC-V for RTOS applications
RVFI	RISC-V Formal Interface
SBI	Supervisor Binary Interface
SDRAM	Synchronous Dynamic Random-Access Memory
SEE	Supervisor Execution Environment

Abbreviation	Definition
SIM	Simulation
SoC	System-on-Chip
TCM	Tightly-Coupled Memory
UART	Universal Asynchronous Receiver Transmitter
WARL	Write Any Values, Reads Legal Values
WDT	Watchdog Timer Device
WFI	Wait for Interrupt

1. Introduction

The Lattice Semiconductor RISC-V RX soft IP contains a 32-bit RISC-V processor core and several submodules – Platform Level Interrupt Controller (PLIC), Core Local Interrupter (CLINT), and Watchdog. The CPU core supports the RV32IMACF instruction set and the debug feature which is JTAG – IEEE 1149.1 compliant. The modules outside are accessed by the processor core using the Advanced Extensible Interface (AXI) or Local Bus Interface.

The design is implemented in Verilog HDL. It can be configured and generated using the Lattice Propel™ Builder software. It is targeted for Certus™-N2, Lattice Avant™, MachXO5™-NX, CrossLink™-NX, Certus-NX, and CertusPro™-NX FPGA devices. The design is implemented using Lattice Radiant™ software Place and Route tool integrated with the Synplify Pro® synthesis tool.

1.1. Quick Facts

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

Table 1.1 RISC-V RX Soft IP Quick Facts

IP Requirements	Supported FPGA Family	Certus-N2, Lattice Avant, MachXO5-NX, CrossLink-NX, Certus-NX, CertusPro-NX
Resource Utilization	Targeted Devices	LN2-CT, LAV-AT, LFMXO5, LIFCL, LFD2NX, LFCPNX
	Supported User Interfaces	AXI, Local Bus Interface, Composable Extension Unit Logic Interface (CXU-LI), RISC-V Formal Interface (RVFI)
	Resources	See Table A.1 and Table A.2.
Design Tool Support	IP Implementation	IP v2.5.0 – Lattice Propel Builder 2024.2, Lattice Radiant™ 2024.2
	Simulation	For a list of supported simulators, see the Lattice Radiant and Lattice Diamond™ software user guide.

1.2. Features

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

- RV32IMACF instruction set
- Five-stage pipeline
- Three privilege modes supported: Machine mode, Supervisor mode, and User mode
- Three processor modes supported: Advanced mode, Balanced mode, and Lite mode
- Instruction Cache and Data Cache
- Debug through Gnu Debugger (GDB) and Open On-Chip Debugger (OpenOCD)
- PLIC module
- CLINT module
- Watchdog module
- Supports AXI, Local Bus Interface, CXU-LI, and RVFI.
- Supports dynamic branch target prediction.
- Supports physical memory protection (PMP).
- Supports soft reset.
- Benchmark and Frequency:
  - Balanced mode: 1.21 DMIPS/MHz performance; 130 MHz (sp9)/110 MHz (sp7) on a CertusPro-NX device
    Note: f_{max} is based on:
    - Standalone processor core
    - Radiant 2023.1 production build, with 9_High-Performance_1.0V (sp9) and 7_High[1]Performance_1.0V (sp7)

1.3. Conventions

1.3.1. Nomenclature

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

1.3.2. Signal Names

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

1.4. Licensing and Ordering Information

The RX 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.

2. Functional Descriptions

2.1. Overview

The RISC-V RX 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 submodules. Among submodules, PLIC and CLINT/Watchdog are required, while Local UART is optional. The AXI instruction port and both TCM ports are also optional.

The 32-bit processor can use the AXI instruction port or the local instruction port to fetch instructions from an external AXI device or a TCM, respectively. The processor can use the AXI data port or the local data port to access data. Among these AXI and local bus ports, the AXI instruction port and both TCM local bus ports, as shown in Figure 2.1, are optional in the RX configuration dialog. But either the AXI instruction port or both of the TCM ports must be enabled to make the RX core perform normally.

The CPU core, bridges, MUX, PLIC, and UART run in the fast system clock domain. The CLINT and the Watchdog run in both the fast system clock domain and the slow real time clock domain. The Debug module runs in both the system clock domain and the JTAG clock domain.

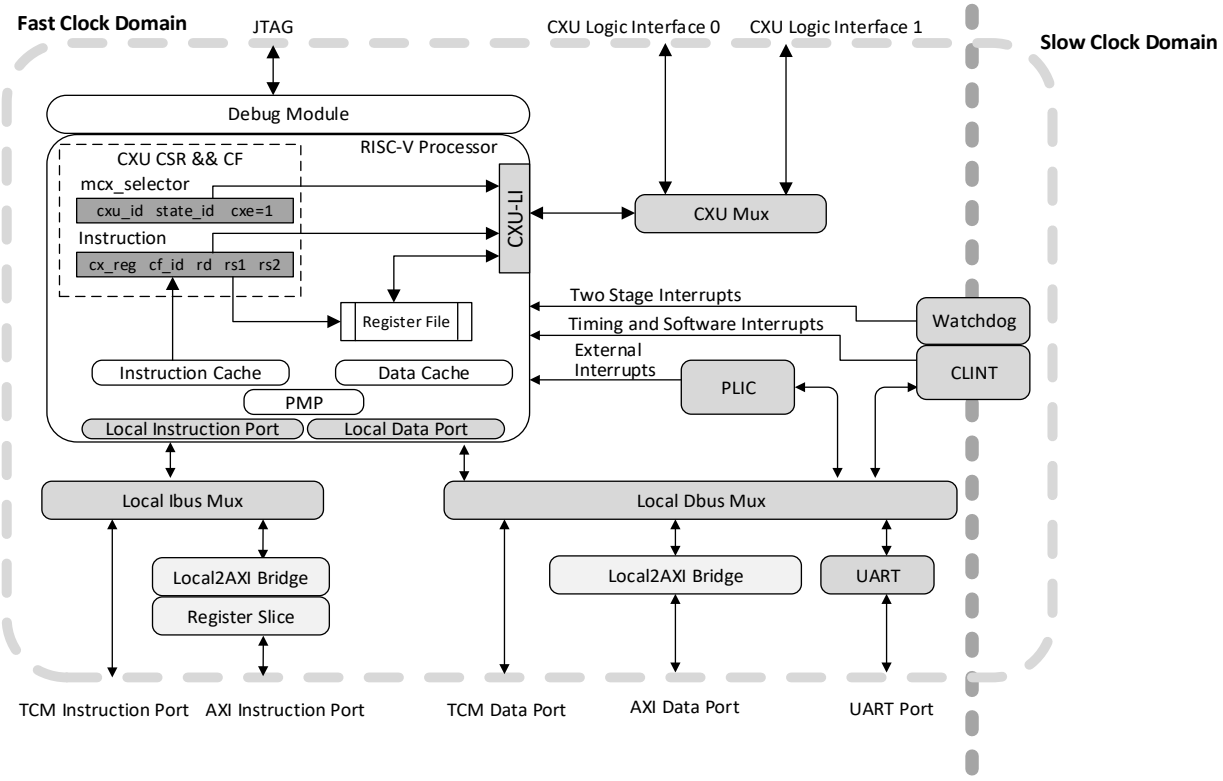


Figure 2.1. RISC-V RX Soft IP Diagram, with All Features Enabled

2.2. Modules Description

2.2.1. RISC-V Processor Core

Figure 2.2 shows the processor core block diagram.

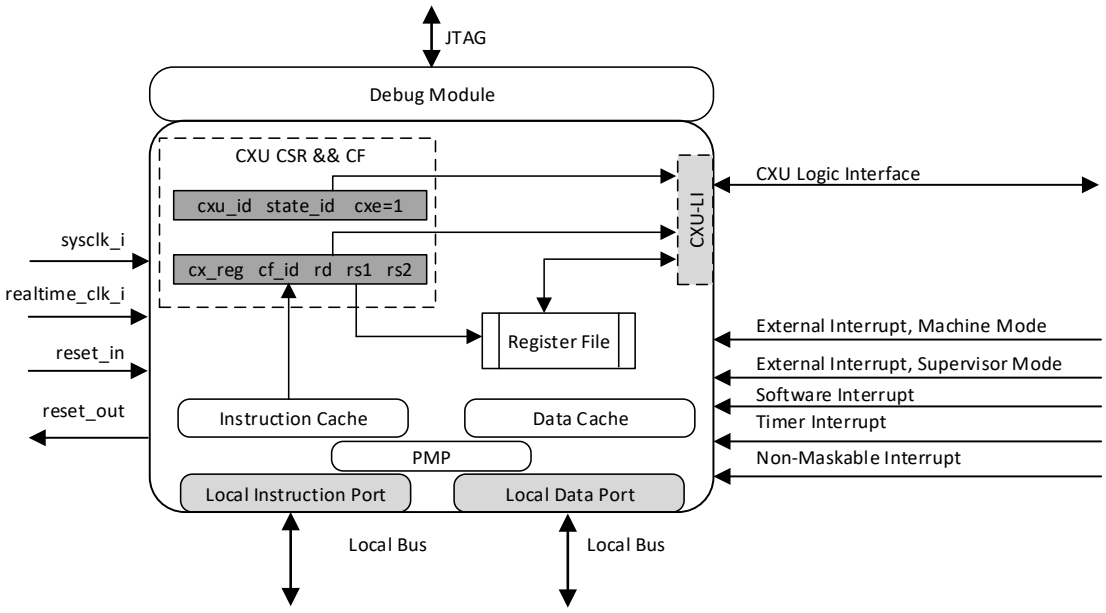


Figure 2.2. RISC-V RX Processor Core Block Diagram

2.2.1.1. Processor Modes

Version 2.5.0 of the RX core supports three processor modes, Lite, Balanced, and Advanced. The Lite mode is designed for smaller areas. The Balanced mode is designed for the balance of performance and resource utilization. The Advanced mode supports all features of the RX core. The three modes have the same configuration for CSR, submodule, and interface. The detailed differences between them are shown in Table 2.1.

Table 2.1. Processor Modes

Mode	Lite	Balanced	Advanced
Misa value	0x224	0x141145	0x141165
Extension	IMC	IMAC	IMACF
I Cache	Not supported	Supported	Supported
D Cache	Not supported	Supported	Supported
Soft Reset	Not supported	Supported	Supported
Configurable Reset Vector	Not supported	Supported	Supported
PMP	Not supported	Supported	Supported

You can select the processor mode through the general tab of Module/IP block wizard GUI as needed (Figure 2.3).

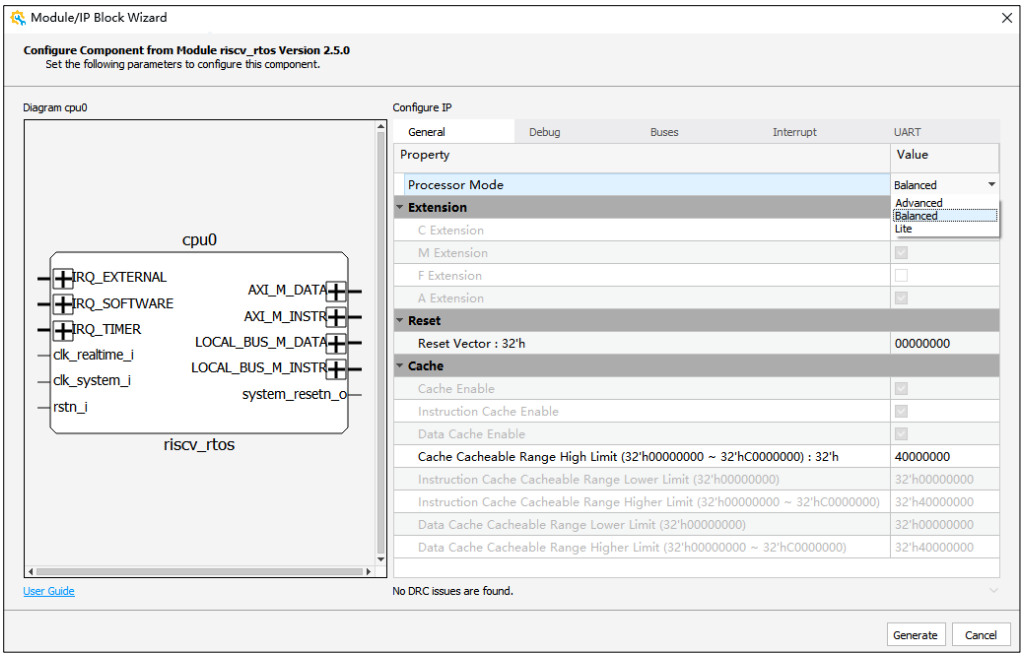


Figure 2.3. Select Processor Mode

2.2.1.2. A Extension Support

The RX core in the Balanced mode and Advanced mode supports the A extension. For more details, refer to the related chapter of RISC-V Instruction Set Manual Volume I: Unprivileged ISA (Version 20191213).

The A Extension is only supported when TCM is enabled. The address accessed by A Extension instructions is only legal in the address space of the TCM. To support A Extension, the TCM IP version must be updated to 1.4.0 or later and the ATOMIC checkbox must be checked.

2.2.1.3. F Extension Support

The RX core in the Advanced mode supports the F extension. For more details, refer to the related chapter of RISC-V Instruction Set Manual Volume I: Unprivileged ISA (Version 20191213).

2.2.1.4. Control and Status Registers

The processor core supports three privilege modes. All supported Control and Status Registers (CSRs) are listed in Table 2.2.

Table 2.2. RISC-V Processor Core Control and Status Registers

Number	Privilege	Name	Description
Supervisor Trap Setup			
0x100	SRW	sstatus	Supervisor status register
0x104	SRW	sie	Supervisor interrupt enable register
0x105	SRW	stvec	Supervisor trap handler base address
0x106	SRW	scounter	Supervisor counter-enable register
Supervisor Trap Handling			
0x140	SRW	sscratch	Scratch register for supervisor trap handlers
0x141	SRW	sepc	Supervisor exception program counter
0x142	SRW	scause	Supervisor trap cause
0x143	SRW	stval	Supervisor bad address or instruction
0x144	SRW	sip	Supervisor interrupt pending

Number	Privilege	Name	Description
Machine Information Registers			
0xF11	MRO	mvendorid	Vendor ID
0xF12	MRO	marchid	Architecture ID
0xF13	MRO	mimpid	Implementation ID
0xF14	MRO	mhartid	Hardware thread ID
Machine Trap Setup			
0x300	MRW	mstatus	Machine status register
0x301	MRO	misa	ISA and extensions
0x302	MRW	medeleg	Machine exception delegation register
0x303	MRW	mideleg	Machine interrupt delegation register
0x304	MRW	mie	Machine interrupt enable register
0x305	MRW	mtvec	Machine trap handler base address
0x306	MRW	mcounteren	Machine counter-enable register
Machine Trap Handling			
0x340	MRW	mscratch	Scratch register for machine trap handlers
0x341	MRW	mepc	Machine exception program counter
0x342	MRO	mcause	Machine trap cause
0x343	MRO	mtval	Machine bad address or instruction
0x344	MRW	mip	Machine interrupt pending
Machine Counter/Timers			
0xB00	MRW	mcycle	Machine cycle counter
0xB02	MRW	minstret	Machine instructions-retired counter
0xB80	MRW	mcycleh	Upper 32 bits of mcycle
0xB82	MRW	minstreth	Upper 32 bits of minstret
PMP			
0x3A0	MRW	pmpcfg0	PMP config register 0
0X3B0	MRW	pmpaddr0	PMP address register 0
0X3B1	MRW	pmpaddr1	PMP address register 1
0X3B2	MRW	pmpaddr2	PMP address register 2
0X3B3	MRW	pmpaddr3	PMP address register 3
Unprivileged Floating-Point CSRs			
0x001	URW	fflags	Floating-point accrued exceptions
0x002	URW	frm	Floating-point dynamic rounding mode
0x003	URW	fcsr	Floating-point control and status register, combining frm and fflags

2.2.1.5. Privilege Modes

The processor supports the User, Supervisor, and Machine mode. Along with corresponding CSR registers. Figure 2.4 shows two typical software stacks:

- A simple system that supports only a single application running on an application execution environment (AEE). The application is coded to run with a particular application binary interface (ABI). ABI includes the supported user-level Instruction Set Architecture (ISA) plus a set of ABI calls to interact with the AEE. The ABI hides details of the AEE from the application to allow greater flexibility in implementing the AEE.
- Meanwhile, a conventional operating system (OS) can provide AEE and ABI. The OS interfaces with a supervisor execution environment (SEE) via a supervisor binary interface (SBI). An SBI comprises the user-level and supervisor-level ISA together with a set of SBI function calls.

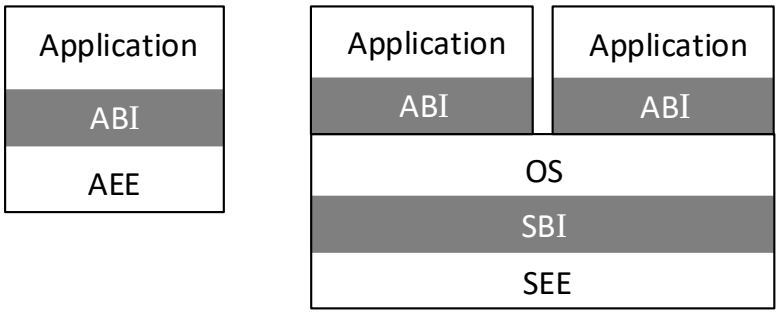


Figure 2.4. Various Forms of Privileged Execution

2.2.1.6. Interrupt

There are four types of interrupts, the external interrupt from PLIC in the Machine mode or Supervisor mode, the software interrupt, the Timer interrupt from CLINT, and non-maskable interrupt from outside.

- External Interrupt
  - In this version, the RX processor core has 32 external interrupts in total, and 30 of them are available to you.
    Note: 0 is reserved and 1 is fixed to connect the Watchdog module.
- NMI
  - A basic non-maskable interrupt (NMI) is supported in the RX core. There is a CSR named mnvec for you to set a specific trap entry for the NMI routine. Its CSR address is 0x7C0.
  - There is an input port nmiInterrupt for the incoming interrupt. When there is an asserted input, the PC jumps to the address stored in mnvec. For other types of interrupts, it jumps to mtvec. Below is an example asm code:

```
#define CSR_MNVEC              0x7C0
…
la t0, trap_entry_nmi
csrw CSR_MNVEC, t0
```

  - The current NMI implementation is non-recoverable. It is expected for the processor to jump into the correct interrupt service, but there is no guarantee for how it gets returned. General interrupt has the mepc CSR register to store the PC address before jumping to the interrupt, so that when the processor returns from it, mepc is used to restore the previous PC address. NMI does not have such a register. If the NMI service routine executes the return from interrupt instruction (MRET), the result is unpredictable.
    Note: There is a task group aiming to define the recoverable NMI, which is still on track, with no stable draft specification yet. The implementation is Lattice specific.

By default, interrupts are handled in the Machine mode. Considering the Supervisor mode is supported, it is possible to delegate certain interrupts to Supervisor mode.

2.2.1.7. Exception

If an exception occurs, the processor core stops the corresponding instruction, flushes all previous instructions, and waits until the terminated instruction reaches the writeback stage before jumping to the exception service routine.

2.2.1.8. Cache

Only the RX core in the Advanced or Balanced mode has caches. The lower cacheable range limit of Instruction cache and Data cache is fixed at 0x00000000. As shown in Figure 2.5, The cacheable range of both the Instruction cache and Data cache is always the same. The two caches are configurable from 0 to 3 GB by configuring the Cache Cacheable Range High Limit value from 0x00000000 to 0xC0000000. The default upper limit of the cacheable range is 0x40000000. So, the default cacheable range is 1 GB. For details, you can refer to the Memory Map section.

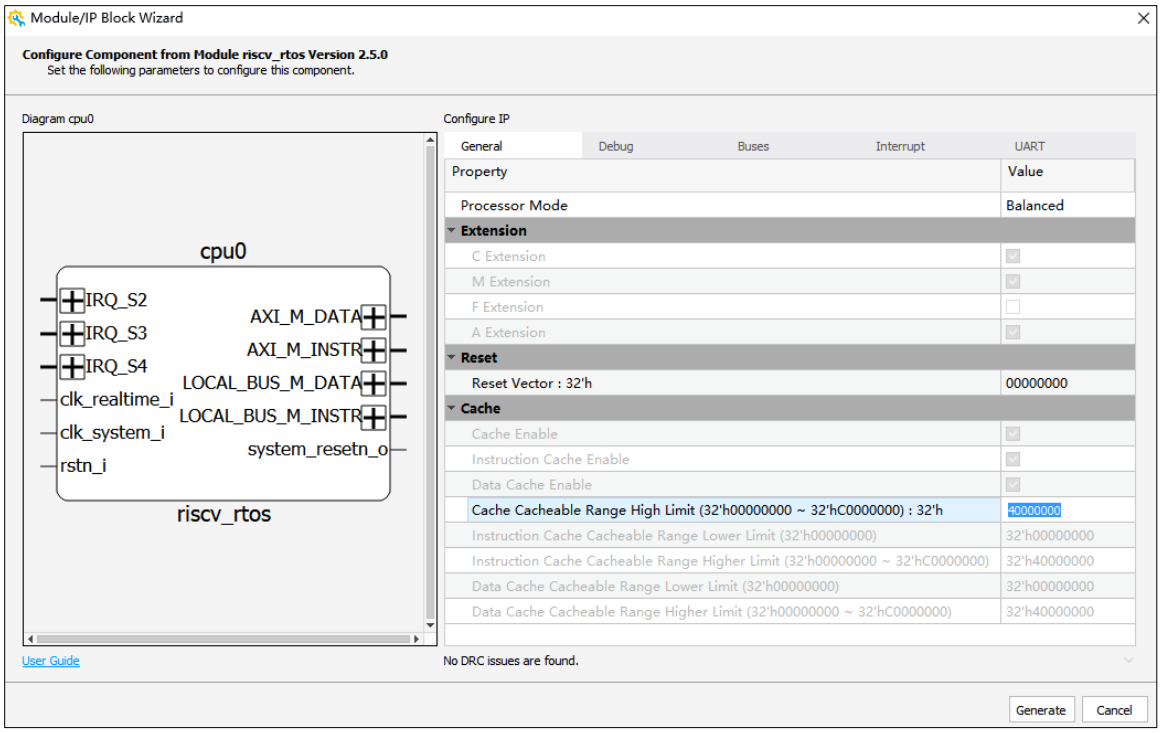


Figure 2.5. Configure Cacheable Range

Both the Instruction cache and Data cache have the following configurations:

- cache size: 4096 bytes
- 32 bytes per cache line
- 2-way set associative

The cache strategy for data cache is write through. The cache eviction policy of both caches is round robin. To flush the caches, refer to annotations of cache.h in the driver codes. There are three APIs to be used to flush either the instruction or data cache at the run time.

2.2.1.9. Low Power Mode

The processor core enters low power mode when it executes the Wait for Interrupt (WFI) instruction. The program counter halts during the low power mode. The processor wakes up if there is an external/timer interrupt.

2.2.1.10. Branch Prediction

The RX core uses dynamic target prediction for branches when cache is enabled and the core is in the Advanced or Balanced mode.

2.2.1.11. Debug

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

You can enable an input port debug_enable to control the debug on/off in run-time (Figure 2.6). Figure 2.7 shows the JTAG types supported.

The JTAG channel is configurable in the IP wizard GUI. The channel range depends on the kind of FPGA device family. For Certus-N2 and Lattice Avant and Nexus family devices, the default JTAG channel range is 14–16. When enabling the Extend JTAG Channel property, 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.

Note: When enabling a larger range of the JTAG channels, the extended JTAG channels may be occupied by another core connected to JTAG, such as Reveal in Lattice platform.

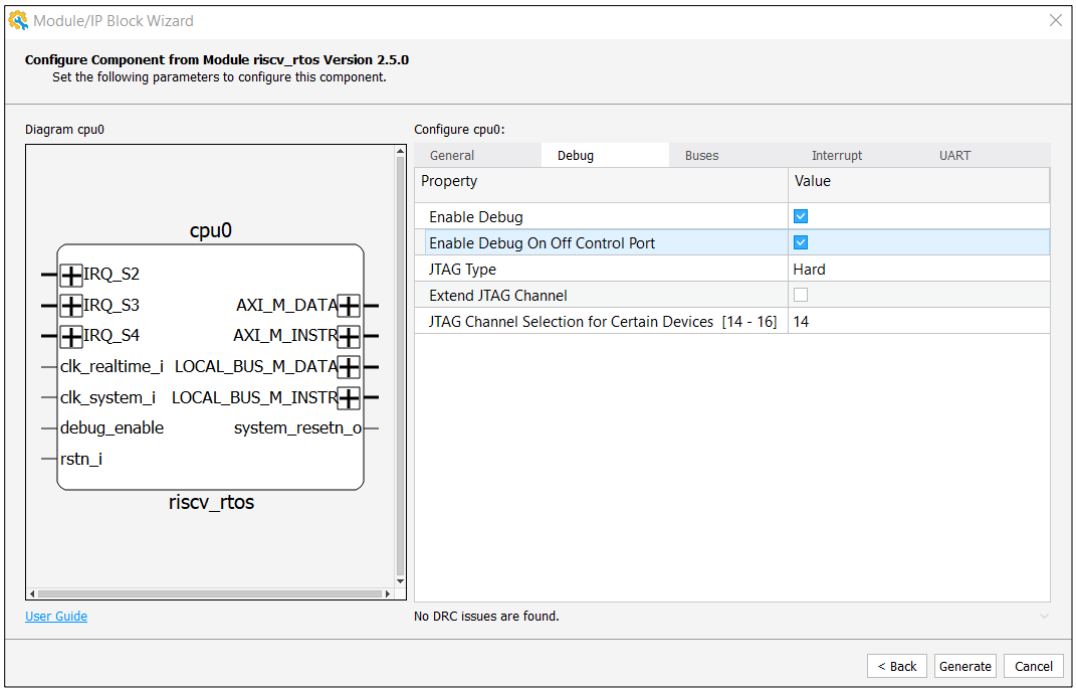


Figure 2.6. Enable Debug On Off Control Port

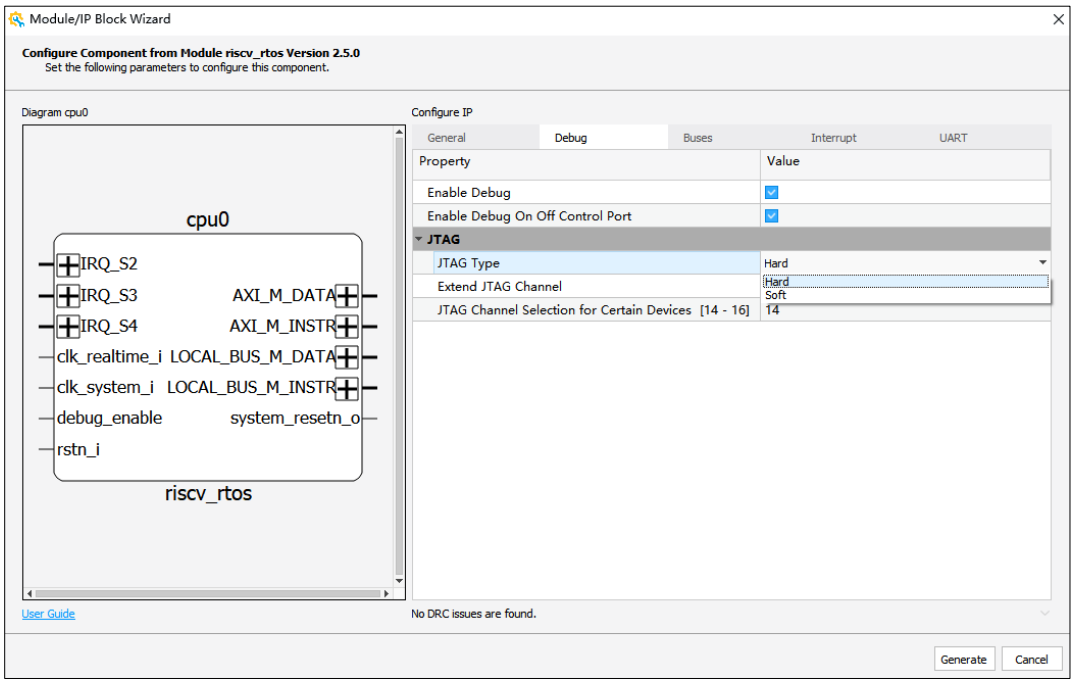


Figure 2.7. JTAG Type

When configuring soft JTAG, the RX core exports a set of JTAG signals. You need to assign FPGA pins manually. For the soft JTAG signals information, refer to the Soft JTAG Interface section. For the soft JTAG ports assigning and corresponding setting information in the Lattice Radiant software, refer to Appendix B.

Note: To use the debug module, it is required to allow writes from the data port to the instruction memory in SoC. The RISC-V core needs to load *.elf image into the program memory and set/unset soft breakpoints through the data port that can write access to the instruction memory.
Single-port instruction memory is not allowed to debug.

2.2.1.12. Physical Memory Protection

The PMP unit provides Machine mode control registers to limit the access of different regions of physical memory with different privileges, including read, write, and execute, for RV32 systems. To support Lattice RISC-V products, the PMP structure only supports the top boundary of an arbitrary range (TOR) mode with up to four entries and the granularity is 0. Our design follows the RISC-V Privileged Specification (Version 1.12).

PMP entries are described by an 8-bit configuration register and one 32-bit address register. These two kinds of registers are packed into CSRs to minimize context-switch time. The PMP configuration registers named pmpcfg# determine the permission and addressing mode for protection regions. The PMP address registers named pmpaddr# contain the address for corresponding regions. # indicates the serial number of register.

This PMP unit partitions the memory range to four pages. There are only four entries for this unit instead of 16 or 64 entries as in the RISC-V specification. In other words, in this PMP unit, there is only one PMP configuration register, pmpcfg0, and four PMP address registers, pmpaddr0–pmpaddr3. All the register fields are WARL registers.

- PMP Configuration Registers

Each pmpcfg# register contains four, 8-bits pmp#cfg register fields to describe the access privileges corresponding to four pmpaddr# for the RV32 system. As mentioned above, only pmpcfg0 is used in this unit and its associated number in CSRs is 0x3A0, as shown in Figure 2.8.

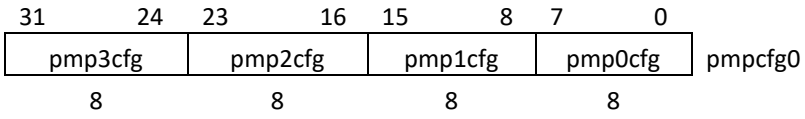


Figure 2.8. RV32 PMP Configuration CSR Layout

Table 2.3 shows the layout of one pmp#cfg register inside pmpcfg0.

Table 2.3. pmp#cfg Register Format

Field	Name	Access	Width	Description
[7]	L	WARL	1	The PMP entry is locked.
[6:5]	0	WARL	2	—
[4:3]	A	WARL	2	Encoding the address-matching mode of the associate PMP address register. Value / Mode / Description 0 / OFF / Null region, disabled 1 / TOR / Top of range
[2]	X	WARL	1	When set, PMP entry permits instruction execution. When clear, instruction execution is denied.
[1]	W	WARL	1	When set, PMP entry permits write. When clear, write is denied.
[0]	R	WARL	1	When set, PMP entry permits read. When clear, read is denied.

The R, W, and X bits determine if this entry allows read, write, or execute respectively.

The A bits encode the address-matching mode. Unlike described in RISC-V Privileged Specification (Version 20211203), this field can only be in two modes, OFF or TOR. The NA4 and NAPOT modes are reserved for future requirements. The L bit indicates whether the entry is locked or not. When the L bit is set, writes to the configuration register and related address registers are ignored. Locked PMP entries are unlocked when the hart is reset. For instance, if the entry i is locked, writes to pmpicfg and pmpaddri are ignored. Additionally, in TOR mode, writing to pmpaddri-1 is also ignored.

- PMP Address Registers

Each pmpaddr# indicates the bits [33:2] of a 34-bits physical address for RV32 systems, as shown in Figure 2.9. Four pmpaddr# are initialized in this unit and their associated numbers in CSRs are 0x3B0 to 0x3B3.

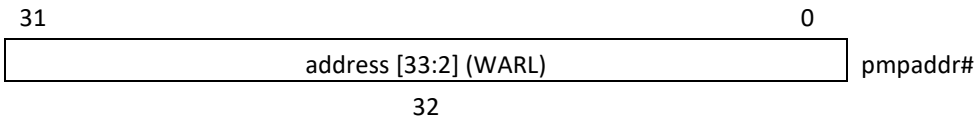


Figure 2.9. PMP Address Register Format, RV32

- Priority and Matching Logics

As shown in Table 2.4, this section describes the logic to verify the access to some region in physical memory. A PMP entry needs to fully match all bytes of an access and then the L, R, W, and X bits determine whether the access passes or fails. If L is clear and the privilege mode is M-Mode, the access succeeds. If L is set, or L is clear with the privilege mode in U-Mode or S-Mode, the access is determined by R, W, and X bits. If no PMP entry matches an M-Mode access, the access succeeds. If no PMP entry matches an S-Mode or U-Mode access, but at least one entry is implemented, the access fails. If at least one access fails, an access-fault exception is generated. The L bit cannot be cleared until the system resets.

Table 2.4. PMP Access Logic

Access Mode	Privilege Mode	Read	Write	Execute
Access in protected range	L = 0 & (M-Mode)	Succeeds		
	L = 0 & (U-Mode || S-Mode)	R bit	W bit	X bit
	L = 1	R bit	W bit	X bit
Access not in protected range	M-Mode	Succeeds		
	U-Mode ||S-Mode	Fails		
Access cross protected and not protected range	Any Mode	Fails		
All entries are off	M-Mode	Succeeds		
All entries are off	U-Mode || S-Mode	Fails		

2.2.1.13. Composable Extension Unit Logic Interface

The Composable Extension Unit Logic Interface 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 controller 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 RX 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 RX 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:

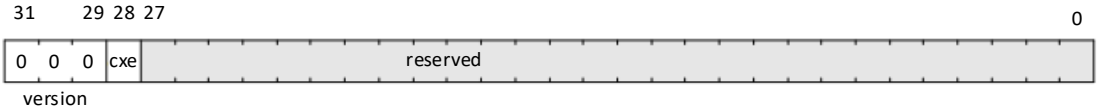


Figure 2.10. mcx_selector CSR 0xBC0 Version 0: Legacy Custom Instructions

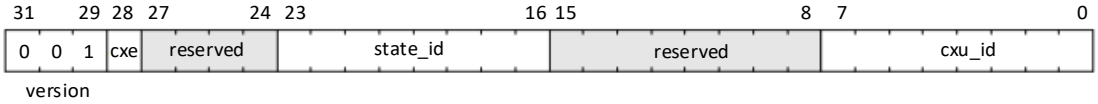


Figure 2.11. mcx_selector CSR 0xBC0 Version 1: Extension Multiplexing

- version: extension multiplexing version
  - 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.
  - 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.
- cxe: custom operation exception enable
  - When version=0 or version=1 and cxe=1, a custom operation raises an illegal-instruction exception.
- state_id: selects the hart's current CXU's current state context.
- cxu_id: selects the hart's current CXU.

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, we reuse three CF formats/major opcodes: 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
     - Assembly instruction: 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.

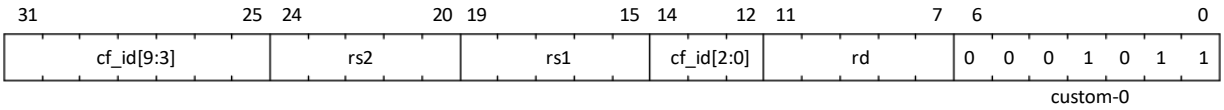


Figure 2.12. CXU R-type Instruction Encoding

- Custom-1 I-type encoding
  - Assembly instruction: 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.

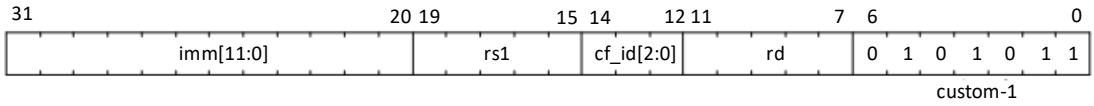


Figure 2.13.CXU I-type Instruction Encoding

- Custom-2 flex-type encoding
  - Assembly instruction: cx_flex cf_id, rs1, rs2
  - Assembly instruction: 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.

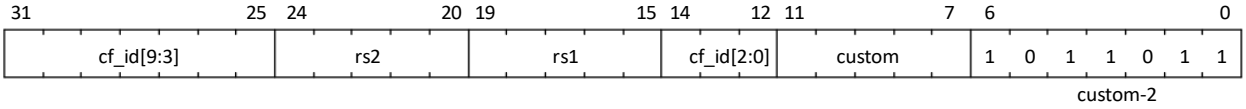


Figure 2.14. CX Flex-type Instruction Encoding

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

Figure 2.15. 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 2.16). 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.

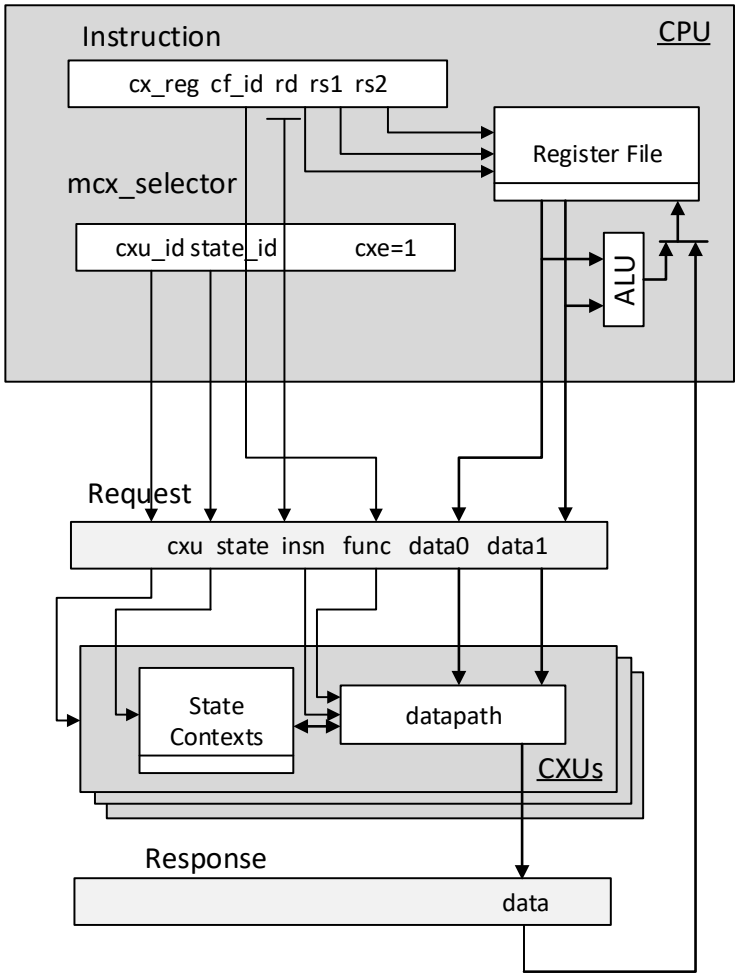


Figure 2.16. Execution of a Custom Function Instruction

Following is a code 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.

2.2.1.14. Set Reset Vector and Soft Reset Built-in Custom Instructions

There are two soft reset related built-in custom instructions supported in the RX core.

One is setting the reset vector. When the new reset vector is loaded to the register identified by the instruction's rs1 region, the CPU issues this instruction subsequently and the reset vector is updated.

31 26	25 20	19 15	14 12	11 7	6 0
101111	reserved	rs1	000	reserved	1111011
					custom-3

Figure 2.17. Custom Instruction Encoding for Setting the Reset Vector

The other one is triggering soft reset.

31 26	25 20	19 15	14 12	11 7	6 0
111111	reserved	rs1	000	reserved	1111011
					custom-3

Figure 2.18. Custom Instruction Encoding for Triggering Soft Reset

The reset range can be controlled by passing a flag value to the instruction specified by rs1. When setting the register value to 0, the reset range only includes the processor core. When setting the register value to 1, the reset range includes the processor core and the submodules. Meanwhile, the reset output port system_resetn_o is pulled down, and the peripheral whose reset connects to it is also reset (Figure 2.19).

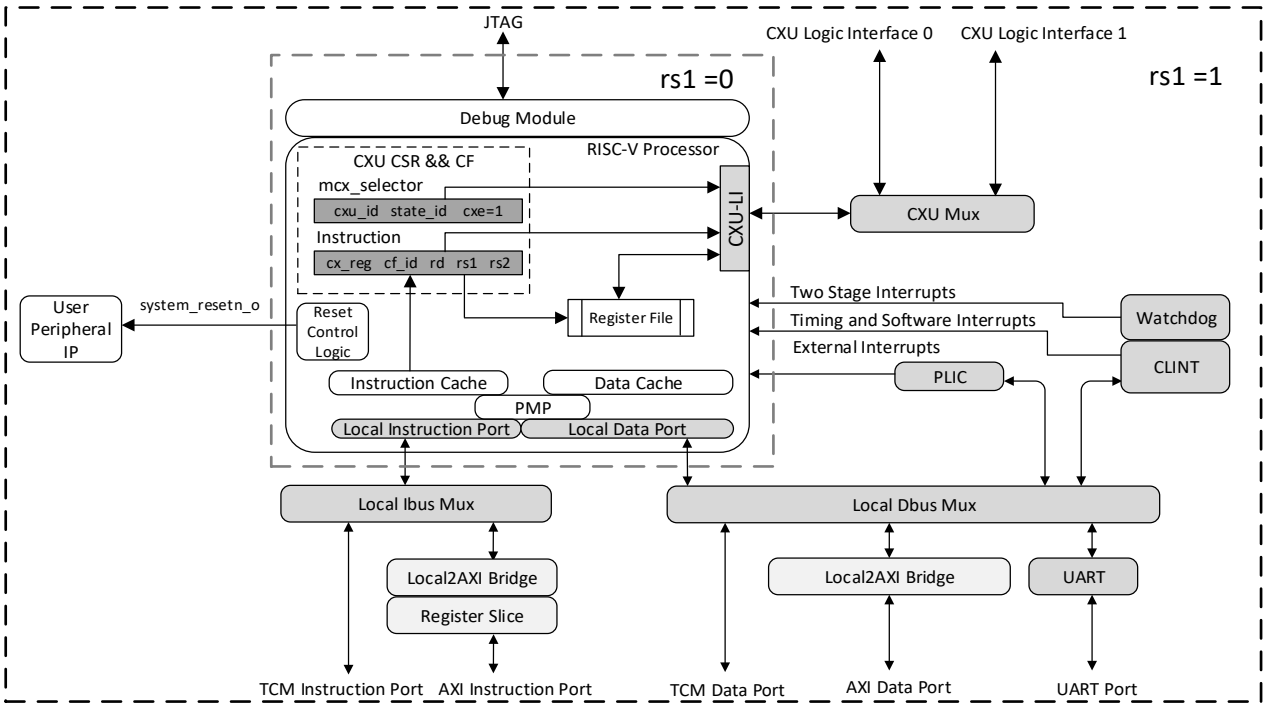


Figure 2.19. Soft Reset Range Control Diagram

The following code shows the scenario of combining the two instructions. A soft reset is executed after a reset vector instruction. When the soft reset is issued, the CPU executes the instruction from the updated reset vector.

```
# define mcx_selector_value 0x00000000 (version=0, cxe=0)
# define func_set_reset_vector 0b111111
# define func_soft_reset 0b101111

lw a5 mcx_selector_value ; load the CSR mcx_selector value for selecting built-in custom
instruction
csrw mcx_selector,a5 ; write the value to CSR mcx_selector
lw a5 reset_vector_value ; load the reset vector value to R1
CUSTOM3, func_set_reset_vector, a5 ; execute setting reset vector
lw a5 reset_range_flag ; load the reset range flag to R1
CUSTOM3, func_soft_reset ,a5 ; execute soft reset
```

Note: When the SoC is reset by the CPU's hardware reset signal rstn_i, the reset vector is reset to the value set in the Module/IP Block Wizard (Figure 2.20).

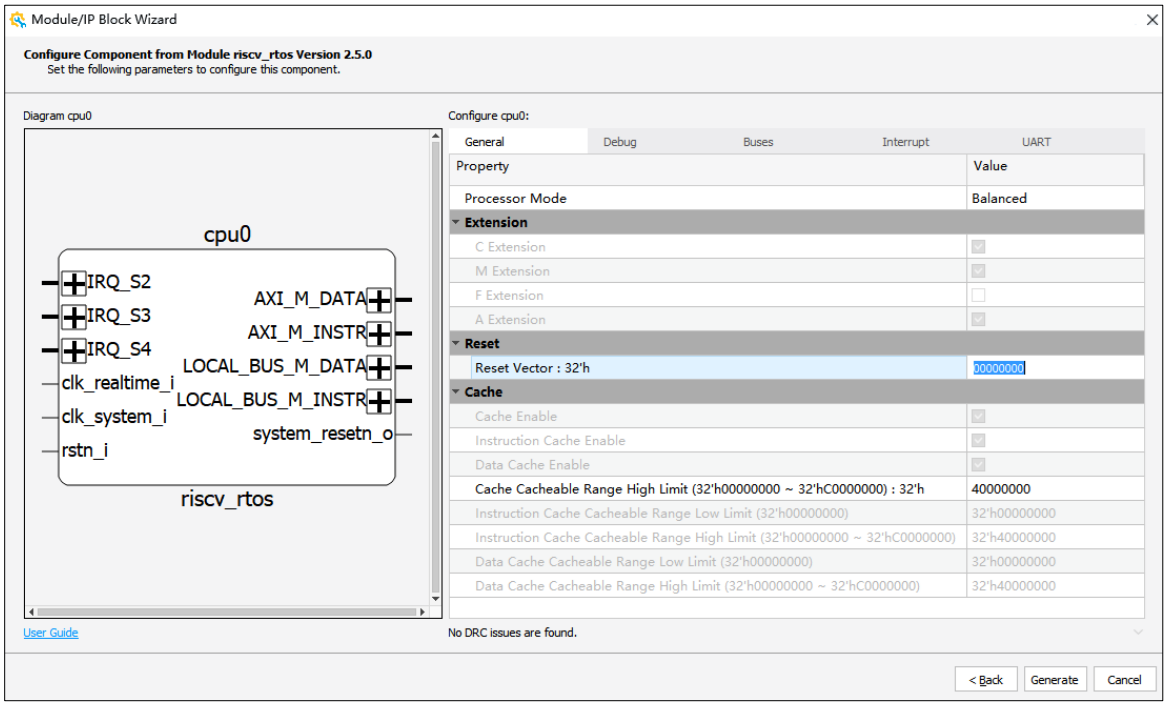


Figure 2.20. Reset Vector Value in Module/IP Block Wizard

2.2.1.15. 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. For signals information, refer to the RVFI Interface section.

2.2.2. Submodules

All the submodules including PLIC, CLINT, and UART are covered in this section. Every submodule has a fixed base address. See Table 2.21 for more details. All the submodules are configurable and scalable through the IP block wizard GUI.

2.2.2.1. Platform Level Interrupt Controller

The PLIC module is compliant with the RISC-V Platform-Level Interrupt Controller Specification (Version 1.0).

The PLIC multiplexes various device interrupts onto the external interrupt lines of Hart contexts, with hardware support for interrupt priorities. The context refers to the specific privilege mode in the specific Hart of specific RISC-V processor instance. PLIC supports up to 31 external interrupts and 0 is reserved. These interrupts are of seven priority levels, and

each one has a corresponding interrupt ID, starting from 1. The first input interrupt (#1) is fixed to Watchdog Timer device.

When PLIC is enabled, IRQ interfaces from 2 to 31 connecting to the PLIC can be exposed upon the configuration in the IP wizard GUI (Figure 2.21). When the IRQ resource and the RX core are in different clock domains, you can add the CDC register to a certain IRQ interface. The CDC register is generated by a two-stage synchronizer. You can enable the CDC register in any enabled IRQ interface.

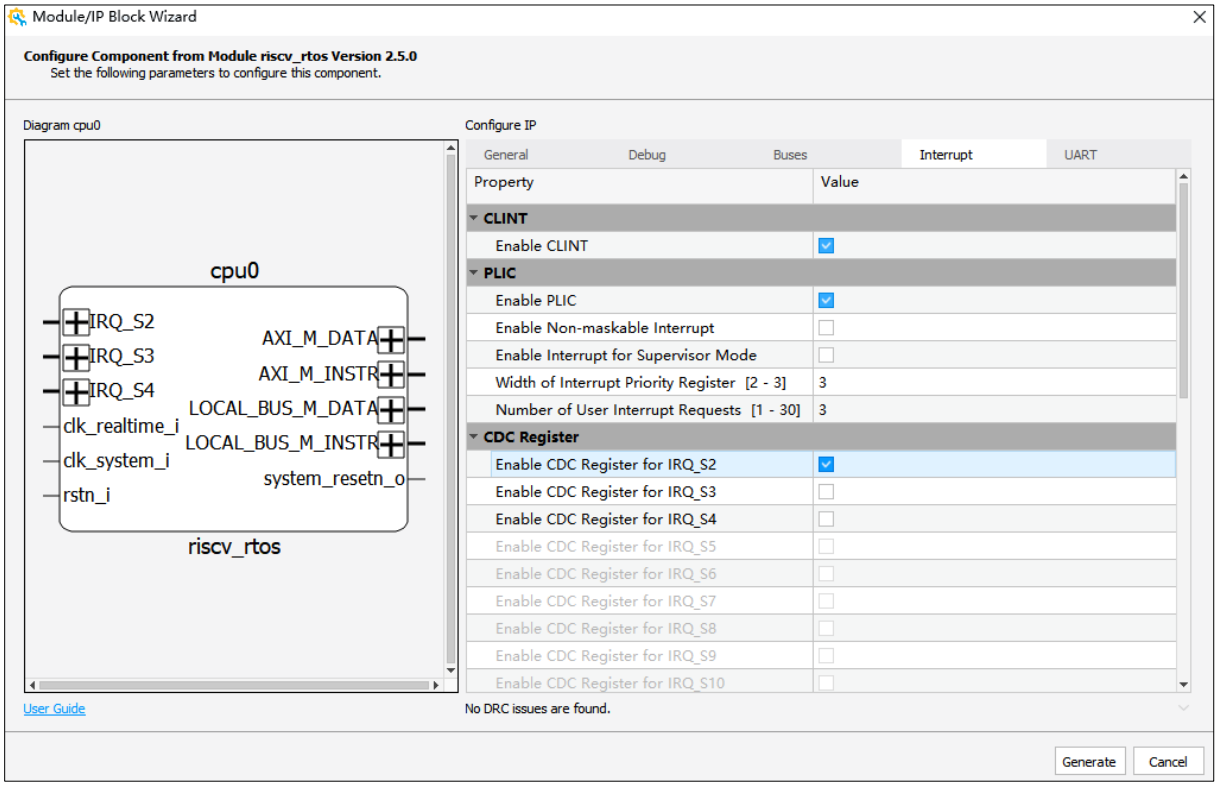


Figure 2.21. Enable CDC Register

When PLIC is disabled, the RX core directly exposes the external interrupt signal IRQ_EXTERNAL outside.

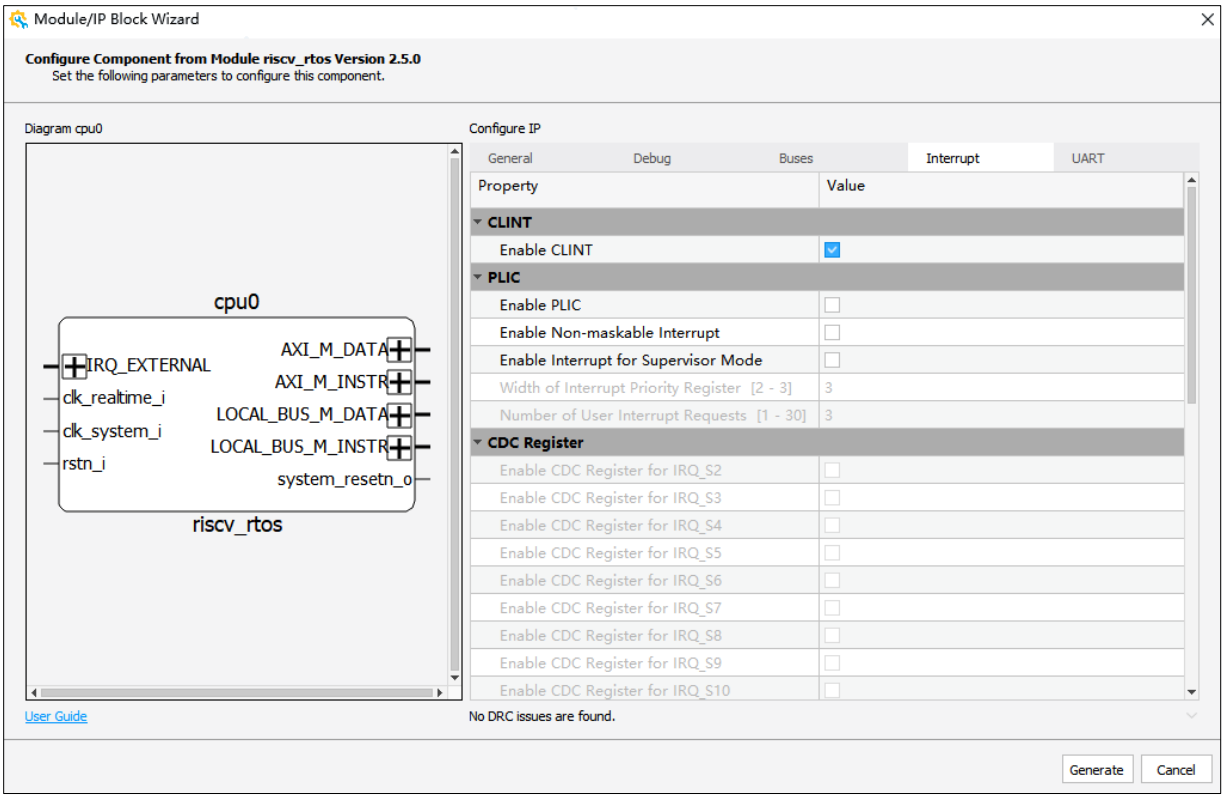


Figure 2.22. Disable PLIC

The PLIC has two interrupt output signals connected to the external interrupt inputs of the CPU – one for the Machine mode, and the other for the Supervisor mode.

Figure 2.23 shows the block diagram of the PLIC operation parameter. An example of how it works: interrupt input 1 gets asserted, it goes through the Gateway and sets the Interrupt Pending bit of the Source. If its Interrupt Enable (IE) is set, the priority value can be passed and compared to other inputs all the way through the chain. The interrupt ID is similarly forwarded. So, if the Max Priority is larger than the threshold, External Interrupt Pending (EIP) can be asserted and sent to the processor. Meanwhile, the Gateway blocks subsequent interrupts from being forwarded until the current interrupt has been completed. Target 0 goes to the Machine mode external interrupt, and Target 1 goes to the Supervisor mode external interrupt.

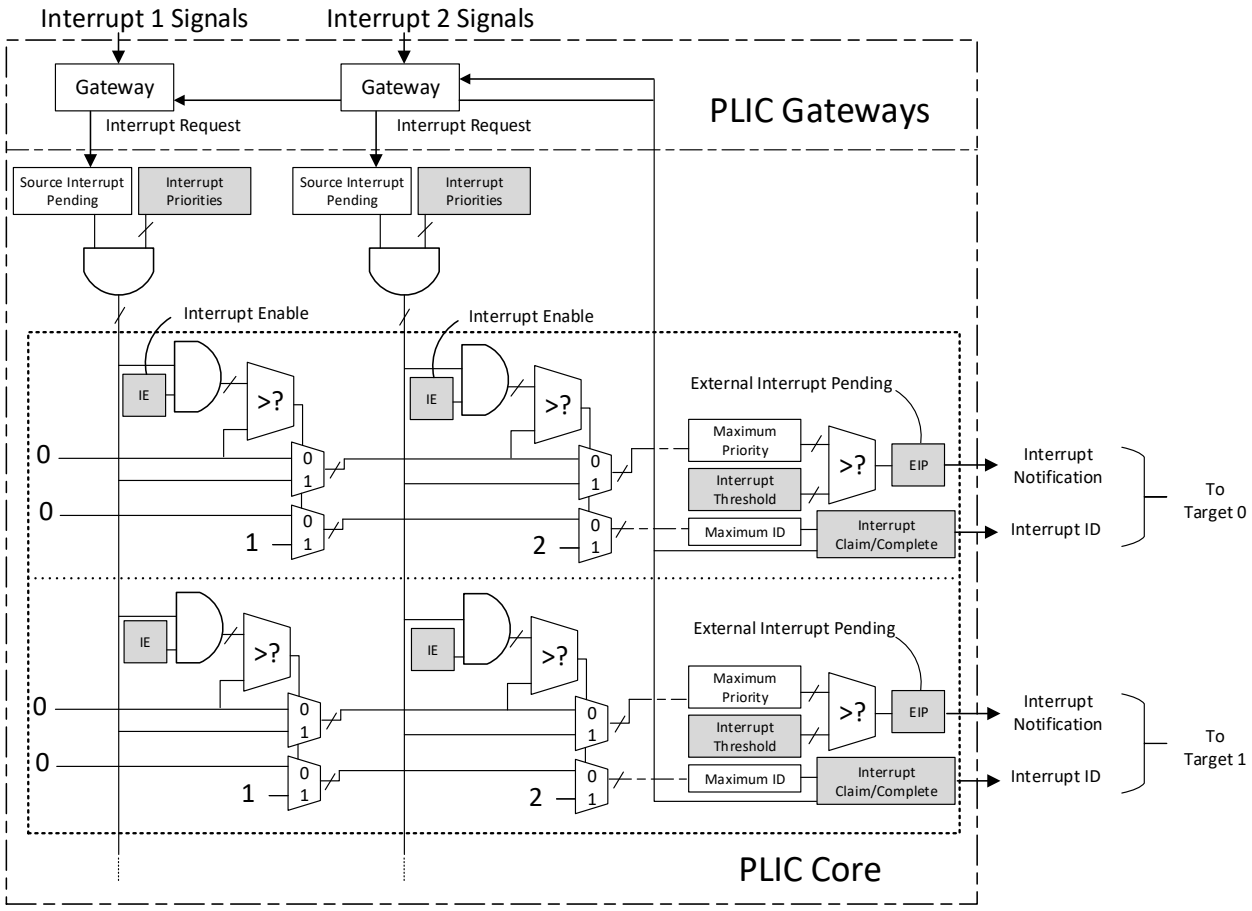


Figure 2.23. PLIC Operation Parameter Block Diagram

The following register blocks are defined in PLIC:

- Interrupt Priorities Registers

Each PLIC interrupt source can be assigned a priority by writing to its 32-bit memory-mapped priority register. A priority value of 0 is reserved to mean never interrupt and effectively disables the interrupt. Priority 1 is the lowest active priority while the maximum level of priority depends on user settings. For example, the highest priority is 3 if the width of the PLIC priority register is set to two. Ties between global interrupts of the same priority are broken by the interrupt ID. Interrupts with the lowest ID have the highest effective priority.

The base address of the interrupt source priority block within the PLIC memory map region is fixed at 0x000000.

- Interrupt Pending Bits Registers

The current status of the interrupt source pending bits in the PLIC core can be read from the pending array, organized as 32-bit registers. The pending bit for interrupt ID N is stored in bit N. Bit 0 of word 0, which represents the non-existent interrupt source 0, is hardwired to zero.

A pending bit in the PLIC core can be cleared by setting the associated enable bit then performing a claim.

The base address of the interrupt pending bits block within the PLIC memory map region is fixed at 0x001000.

- Interrupt Enables Registers

Each global interrupt can be enabled by setting the corresponding bit in the enables registers. The enables registers are accessed as a contiguous array of 32-bit registers, packed the same way as the pending bits. Bit 0 of enable register 0 represents the non-existent interrupt ID 0 and is hardwired to 0. PLIC has two interrupt enable blocks, one for each context.

The context refers to the specific privilege mode in the specific Hart of specific RISC-V processor instance.

For the current IP, context 0 refers to hart 0 of the Machine mode and context 1 refers to hart 0 of the Supervisor mode.

The base address of the interrupt enable bits block within the PLIC memory map region is fixed at 0x002000.

- Priority Thresholds Registers

PLIC provides context-based threshold register for the settings of an interrupt priority threshold of each context. The threshold register is a WARL field. The PLIC masks all PLIC interrupts of a priority less than or equal to threshold. For example, a threshold value of zero permits all interrupts with non-zero priority.

The base address of the priority thresholds registers block is located at 4K alignment starting from offset 0x200000.

- Interrupt Claim Registers

The PLIC can perform an interrupt claim by reading the claim/complete registers, which return the ID of the highest priority pending interrupt or zero if there is no pending interrupt. A successful claim also atomically clears the corresponding pending bit on the interrupt source.

The PLIC can perform a claim at any time and the claim operation is not affected by the setting of the priority thresholds registers.

The interrupt claim register is context-based and is located at 4K alignment + 4 starting from offset

0x200000.

- Interrupt Completion Registers

The PLIC signals the completion of executing an interrupt handler by the host signaling the PLIC and writing the interrupt ID received from the claim to the claim/complete register. The PLIC does not check whether or not the completion ID is the same as the last claim ID for that target. If the completion ID does not match an interrupt source that is currently enabled for the target, the completion is silently ignored.

The interrupt completion registers are context-based and located at the same address as the interrupt claim register, which is at 4K alignment + 4 starting from offset 0x200000.

Table 2.5 provides the description of PLIC registers.

Table 2.5. PLIC Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x00_0000</td><td>—</td><td>Reserved
Interrupt source 0 does not exist.</td></tr>
<tr><td>0x00_0004</td><td>PLIC_PRIORITY_SRC1</td><td>Interrupt Source 1 Priority
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:3]</td><td>Reserved</td><td>RO</td><td>29</td><td>0x0</td></tr>
<tr><td>[2:0]</td><td>Priority</td><td>RW</td><td>3</td><td>0x0</td></tr>
</table>
Priority:
Sets the priority for a given global interrupt.</td></tr>
<tr><td>0x00_0008
....
0x00_007C</td><td>PLIC_PRIORITY_SRC2
...
PLIC_PRIORITY_SRC31</td><td>Same as PLIC_PRIORITY_SRC1.</td></tr>
<tr><td>...</td><td>—</td><td>—</td></tr>
<tr><td>0x00_1000</td><td>PLIC_PENDING1</td><td>PLIC Interrupt Pending Register 1
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:1]</td><td>PendingN</td><td>RO</td><td>31</td><td>0x0</td></tr>
<tr><td>[0]</td><td>Pending0</td><td>RO</td><td>1</td><td>0x0</td></tr>
</table>
Pending0:
Non-existent global interrupt 0 is hardwired to zero.
PendingN:
Equal to PLIC_PENDING1[N], pending bit for global interrupt N.</td></tr>
<tr><td>...</td><td>—</td><td>—</td></tr>
<tr><td>0x00_2000</td><td>PLIC_ENABLE1_M</td><td>PLIC Interrupt Enable Register 1 for Hart 0 M-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:1]</td><td>EnableN</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>Enable0</td><td>RO</td><td>1</td><td>0x0</td></tr>
</table>
Enable0:
Non-existent global interrupt 0 is hardwired to zero.
EnableN:
Equal to PLIC_ENABLE_M[N], enable bit for global interrupt N.</td></tr>
<tr><td>...</td><td>—</td><td>—</td></tr>
<tr><td>0x00_2080</td><td>PLIC_ENABLE1_S</td><td>PLIC Interrupt Enable Register 1 for Hart 0 S-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:1]</td><td>EnableN</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>Enable0</td><td>RO</td><td>1</td><td>0x0</td></tr>
</table>
Enable0:
Non-existent global interrupt 0 is hardwired to zero.
EnableN:
Equal to PLIC_ENABLE_S[N], enable bit for global interrupt N.</td></tr>
<tr><td>...</td><td>—</td><td>—</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x20_0000</td><td>PLIC_THRESHOLD1_M</td><td>PLIC Interrupt Priority Threshold Register for Hart 0 M-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:3]</td><td>Reserved</td><td>RO</td><td>29</td><td>0x0</td></tr>
<tr><td>[2:0]</td><td>Threshold</td><td>RW</td><td>3</td><td>0x0</td></tr>
</table>
Threshold:
Sets the priority threshold.</td></tr>
<tr><td>0x20_0004</td><td>PLIC_CLAIM_1_M</td><td>PLIC Claim Register for Hart 0 M-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>Claim</td><td>RO</td><td>32</td><td>0x0</td></tr>
</table>
Claim:
Read-only field, which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt. A successful claim also atomically clears the corresponding pending bit on the interrupt source.</td></tr>
<tr><td>0x20_0004</td><td>PLIC_COMPLETE_1_M</td><td>PLIC Complete Register for Hart 0 M-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>Completion</td><td>WO</td><td>32</td><td>0x0</td></tr>
</table>
Completion:
Write-only field, write to it to complete the interrupt process.</td></tr>
<tr><td>…</td><td>—</td><td>—</td></tr>
<tr><td>0x20_1000</td><td>PLIC_THRESHOLD1_S</td><td>PLIC Interrupt Priority Threshold Register for Hart 0 S-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:3]</td><td>Reserved</td><td>RO</td><td>29</td><td>0x0</td></tr>
<tr><td>[2:0]</td><td>Threshold</td><td>RW</td><td>3</td><td>0x0</td></tr>
</table>
Threshold:
Sets the priority threshold.</td></tr>
<tr><td>0x20_1004</td><td>PLIC_CLAIM_1_S</td><td>PLIC Claim Register for Hart 0 S-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>Claim</td><td>RO</td><td>32</td><td>0x0</td></tr>
</table>
Claim:
Read-only field, which returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt. A successful claim also atomically clears the corresponding pending bit on the interrupt source.</td></tr>
<tr><td>0x20_1004</td><td>PLIC_COMPLETE_1_S</td><td>PLIC Complete Register for Hart 0 S-Mode
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>Completion</td><td>WO</td><td>32</td><td>0x0</td></tr>
</table>
Completion:
Write-only field, write to it to complete the interrupt process.</td></tr>
</table>

2.2.2.2. Core Local Interrupter

The CLINT module implements mtime, mtimecmp, and some other memory-mapped CSR registers that are associated with timer and software interrupts. When CLINT is disabled, the RX core directly exposes the timer interrupt signal IRQ_TIMER and software interrupt signal IRQ_SOFTWARE outside (Figure 2.24).

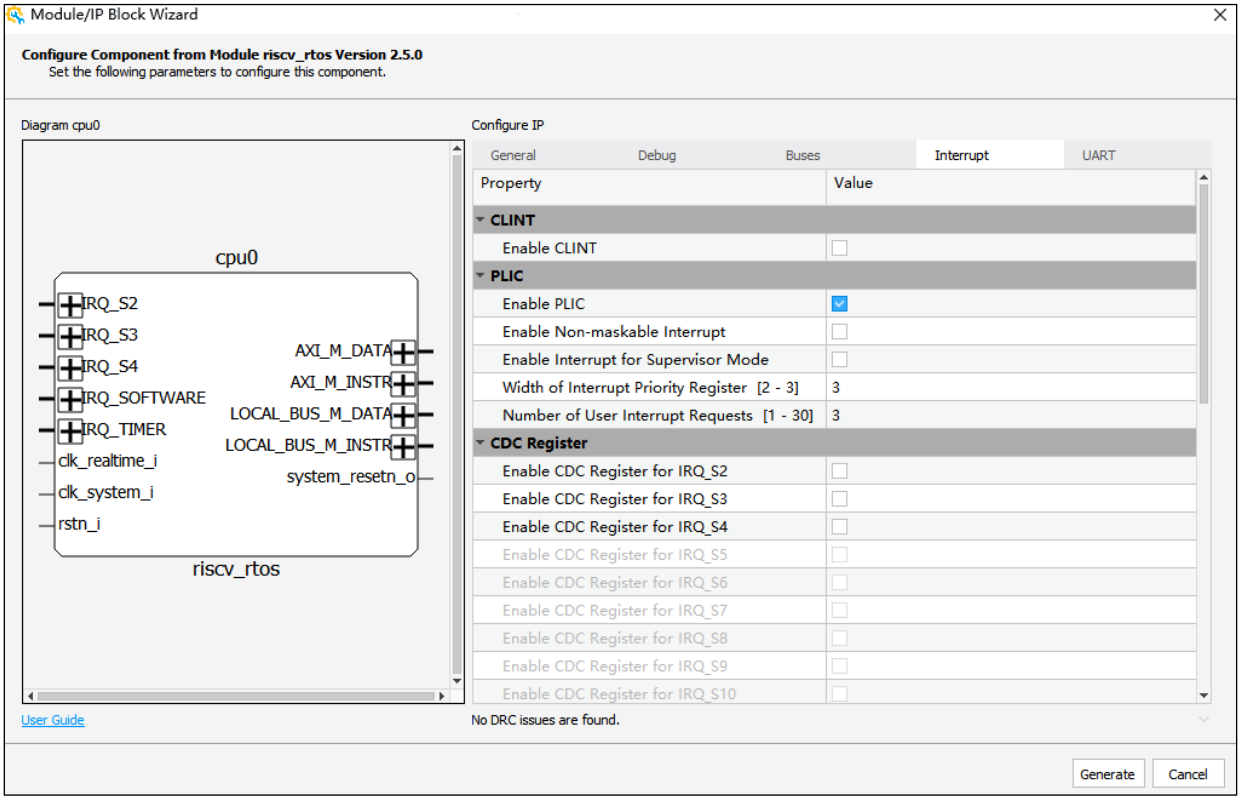


Figure 2.24. Disable CLINT

There are two clocks for CLINT. The msip register is clocked by the system clock, while the mtimecmp and mtime are clocked by a real-time clock which is typically 32 kHz for Lattice FPGAs.

For Certus-N2 and Lattice Avant family devices, you cannot directly get the 32 kHz output from OSC and PLL. Thus, a 512 clock divider is designed for the real-time clock port of the RX core, clk_realtime_i. It is recommended the input of the real-time clock port be a 16.384 MHz clock signal, and it can later provide the 32 kHz clock signal to mtimecmp and mtime registers. It is also legal to set up a personal-defined real-time clock. Note the CLINT_MTIME register is driven by the real-time clock. Therefore, it is required to calculate the timer counter register based on the personal-defined clock. It is safe and recommended to block the real-time and system clock by adding design constraints, such as false_false or set_clock_groups, to avoid any unexpected timing analysis during place and route.

For other family devices, you can directly configure a 32 kHz output on OSC. Then, you can connect this low frequency clock to the real-time clock port.

Table 2.6 provides the descriptions of CLINT registers.

Table 2.6. CLINT Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x00_0000</td><td>CLINT_MSIP</td><td>MSIP Register for hart 0
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:1]</td><td>Reserved</td><td>RO</td><td>31</td><td>0x0</td></tr>
<tr><td>[0]</td><td>msip</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>
msip:</td></tr>
</table>

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td></td><td></td><td>Reflects the memory-mapped MSIP bit of the mip CSR register.
Writing a 1 in the msip field results in the generation of software interrupt.</td></tr>
<tr><td>…</td><td>—</td><td>—</td></tr>
<tr><td>0x00_4000</td><td>CLINT_MTIMECMP_L</td><td>Machine Timer Register – mtimecmp
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>mtimecmp_l</td><td>RW</td><td>32</td><td>Unchanged</td></tr>
</table>
mtimecmp_l:
Lower 32 bits of the mtimecmp CSR register. The first reset value is 0xFFFF_FFFF, after first write, the reset does not change the value of this field.</td></tr>
<tr><td>0x00_4004</td><td>CLINT_MTIMECMP_H</td><td>Machine Timer Register – mtimecmp
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>mtimecmp_h</td><td>RW</td><td>32</td><td>Unchanged</td></tr>
</table>
mtimecmp_h:
Higher 32 bits of the mtimecmp CSR register. The first reset value is 0xFFFF_FFFF, after first write, the reset does not change the value of this field.</td></tr>
<tr><td>…</td><td>—</td><td>—</td></tr>
<tr><td>0x00_BFF8</td><td>CLINT_MTIME_L</td><td>Machine Timer Register – mtime
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>mtime_l</td><td>RW</td><td>32</td><td>0x0</td></tr>
</table>
mtime_l:
Lower 32 bits of the mtime CSR register.</td></tr>
<tr><td>0x00_BFFC</td><td>CLINT_MTIME_H</td><td>Machine Timer Register – mtime
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:0]</td><td>mtime_h</td><td>RW</td><td>32</td><td>0x0</td></tr>
</table>
mtime_h:
Higher 32 bits of the mtime CSR register.</td></tr>
</table>

2.2.2.3. Watchdog Timer

The watchdog timer device (WDT) provides a simple two-stage timer controlled through one memory-mapped CSR register, WDCSR.

WDT waits a software-configured period of time with the expectation that system software re-initializes the watchdog state, reloading the counter by a signal write to WDCSR within this period of time. If this time period elapses without software re-init occurring, then a first-stage timeout register bit S1WTO is set within WDCSR that asserts an interrupt request output signal to notify the system of a stage 1 watchdog timeout. If a second period of time elapses without software re-init of the watchdog, then a second-stage timeout register bit S2WTO is set within WDCSR that generates a system reset.

For current IP, the stage 1 watchdog timeout is connected to PLIC input channel 1 and stage 2 watchdog timeout is connected to system reset.

The mtime CSR Register provides the time base for the watchdog timeout period. The timeout period itself – in units of watchdog clock tick – is specified by the WTOCNT field of the WDCSR CSR register. When WDCSR is written, the WTOCNT value initializes a down counter that decrements with each watchdog tick.

The watchdog tick occurs when bit 14 of mtime transitions from 0 to 1. So the watchdog timeout period is 0.512 second, based on real-time clock of 32 kHz. Meanwhile, the maximum timeout period, WTOCNT = 0x3FF, is about 524 seconds.

WDT is included in the CLINT module. WDT shares the same base address with CLINT, 0xF200_0000. Table 2.7 provides the description of WDT registers.

Table 2.7. WDT Registers

<table>
<tr><th>Offset</th><th>Name</th><th>Description</th></tr>
<tr><td>0x00_D000</td><td>WDT_WDCSR</td><td>Watchdog Register
<table>
<tr><th>Field</th><th>Name</th><th>Access</th><th>Width</th><th>Reset</th></tr>
<tr><td>[31:14]</td><td>Reserved</td><td>RO</td><td>18</td><td>0x0</td></tr>
<tr><td>[13:4]</td><td>WTOCNT</td><td>RW</td><td>10</td><td>0x0</td></tr>
<tr><td>[3]</td><td>S2WTO</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[2]</td><td>S1WTO</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[1]</td><td>Reserved</td><td>RW</td><td>1</td><td>0x0</td></tr>
<tr><td>[0]</td><td>WDEN</td><td>RW</td><td>1</td><td>0x0</td></tr>
</table>
WDEN:
When set, enables the WDT. When clear, the WDT is disabled and S1WTO and S2WTO output signals are forced to be 0, de-asserted. When system reset is asserted, WDT is disabled accordingly by setting WDEN to 0.
S1WTO:
Stage 1 watchdog timeout, active high.
S2WTO:
Stage 2 watchdog timeout, active high.
WTOCNT:
10-bit timeout counter. If it is non-zero and WDEN is set, it decrements every timeout period.</td></tr>
</table>

2.2.2.4. UART

There is an optional fixed memory assignment local UART. When enabling the UART instance, uart_txd_o and uart_rxd_i are exported (Figure 2.25). For related signal information, refer to the UART Ports section.

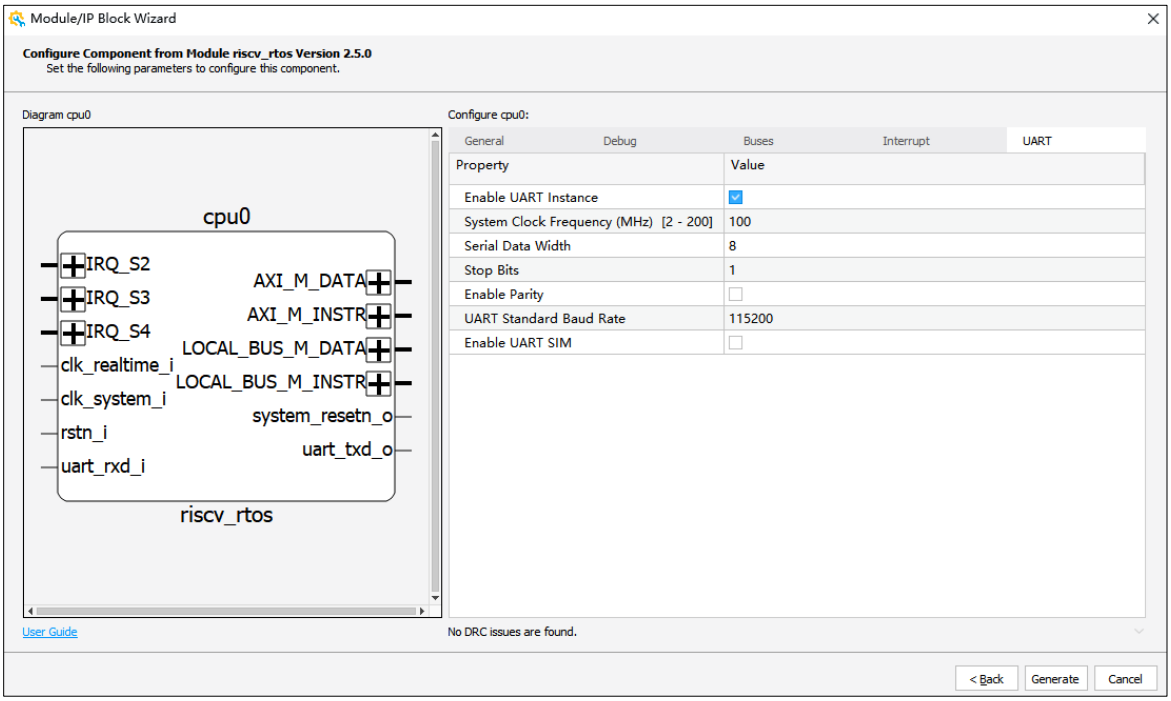


Figure 2.25. Enable UART Ports

2.3. Signal Description

Table 2.8 to Table 2.14 list the ports of the RX CPU soft IP in different categories.

2.3.1. sysClock and Reset

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 a register connecting to the input reset signal rstn_i. It is synchronous with the input clock. When the debugger is enabled and debug reset is issued, the debug reset signal is synchronized to the system clock domain and the system_resetn_o is the output of the synchronized signal.

Table 2.8. Clock and Reset Ports

Name	Direction	Width	Description
clk_system_i	In	1	High speed system clock input
clk_realtime_i	In	1	Low speed real time clock input
rstn_i	In	1	System reset, active low
system_resetn_o	Out	1	Combined system reset and Debug Reset from JTAG

2.3.2. Data Interface

The RX core provides an optional local bus port to connect TCM, while an extra optional AXI interface for the instruction port is available for accessing other memory mapped components such as a flash controller or DDR controller. AXI data ports are fixed. You can edit the AXI ID ports width, the instruction and data ports ID number, and instruction slice type based on request.

The RX core can configure three combinations of the local memory bus and AXI interface in the Module/IP Block Wizard GUI as needed.

The first configuration is only enabling the local bus in the Module/IP Block Wizard GUI (Figure 2.26). The RX core configures the AXI data, local memory bus data, and local memory bus instruction ports. The RX can only fetch program instructions via the local bus instruction interface.

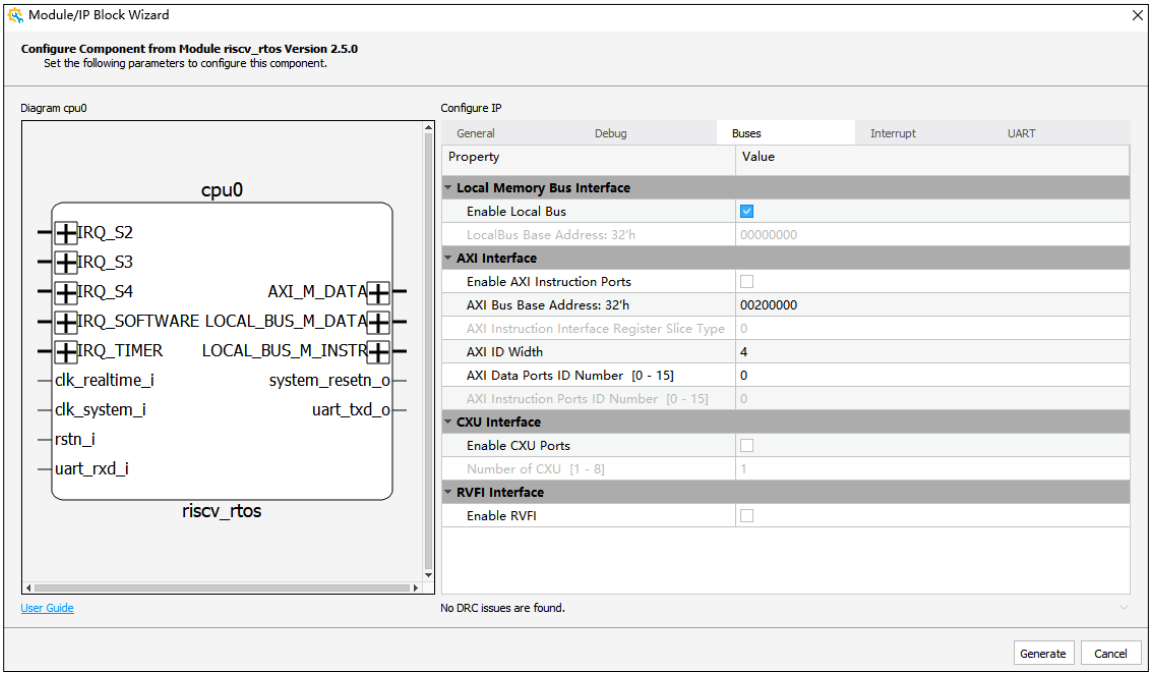


Figure 2.26. Enable Local Bus

In certain scenarios, there is a need to have an exported AXI instruction port. For example, the instruction may come from an external flash through a flash controller. The following two interface combinations can support this scenario.

One is only enabling AXI instruction ports in the Module/IP Block Wizard GUI (Figure 2.27). The RX core configures the AXI interface. The RX core can fetch instructions from memory components like system memory or external DDR memory via the AXI interface.

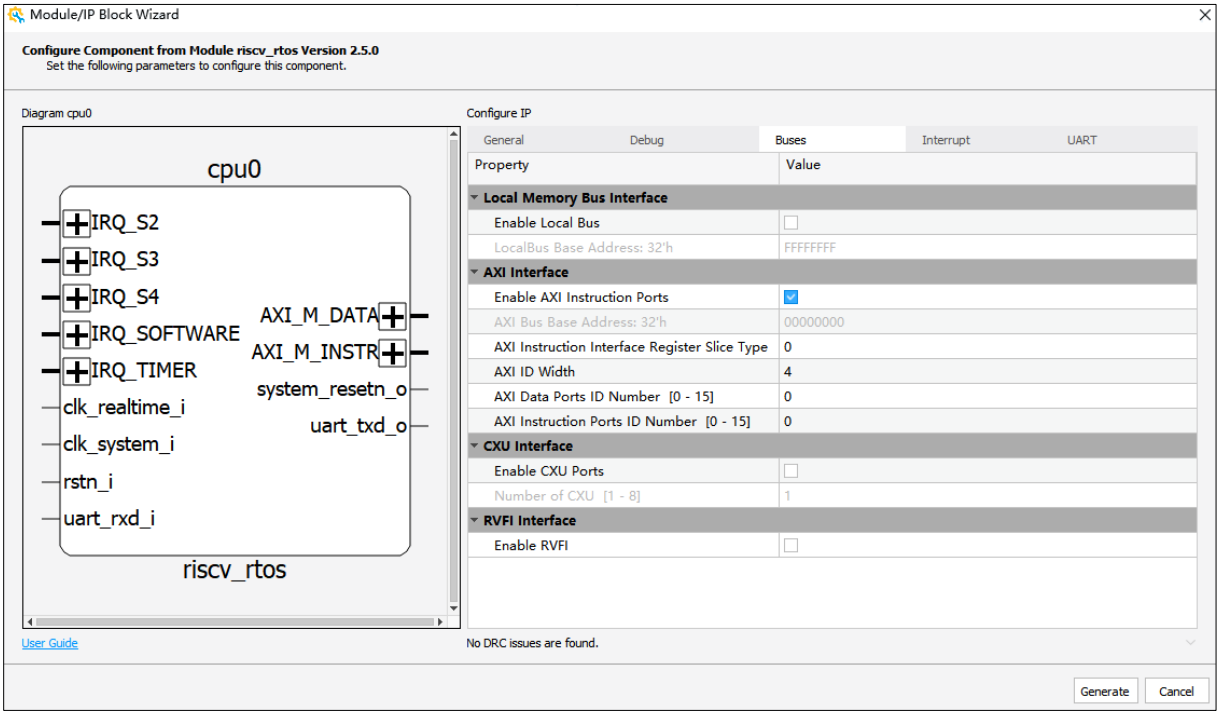


Figure 2.27. Enable AXI Instruction Ports

The other option is enabling the local bus and AXI instruction ports at the same time (Figure 2.28). The RX core configures AXI data, local memory bus data, and local memory bus instruction ports. The RX core can access a program memory file stored in TCM through the local memory bus interface and the other program memory file stored in external memory components through the AXI interface.

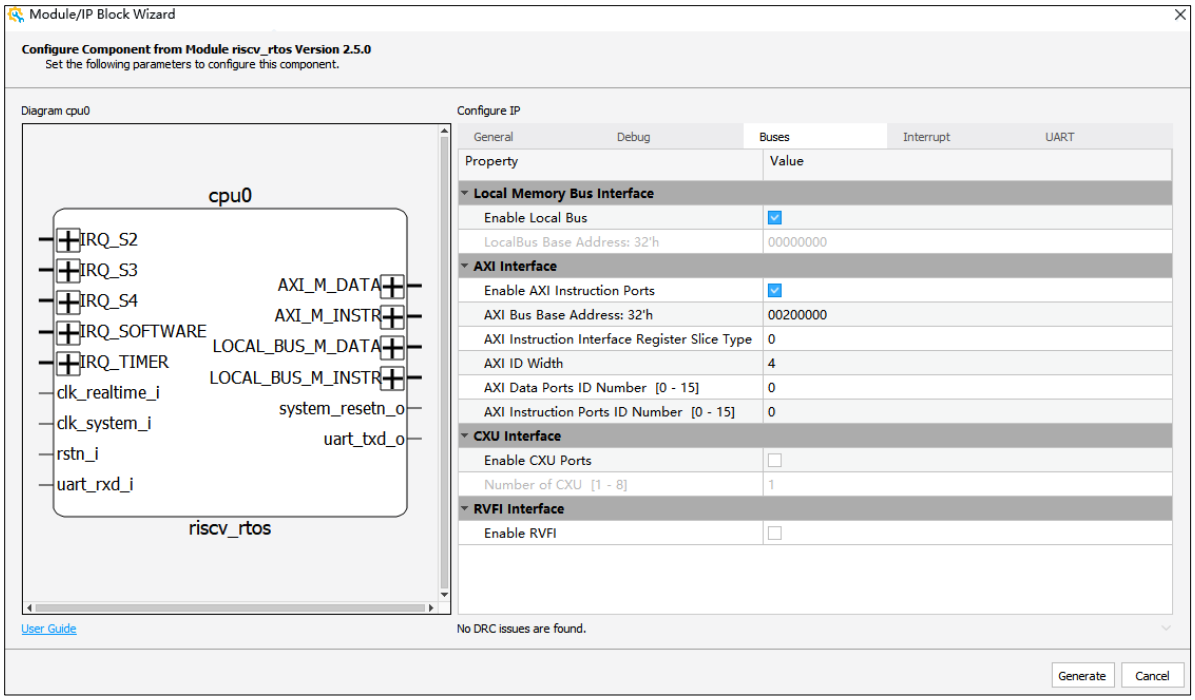


Figure 2.28. Enable Local Bus and AXI Instruction Ports

The RX core supports the write response. A write error on the local and AXI bus on the processor causes the Store/AMO access fault exception of the core, exception ID: 7.

Meanwhile, to remove potential dependency on other components at SoC level, there is an option for you to enable register slice for AXI-based instruction or data port, as shown in Figure 2.1.

Table 2.9. Local Data Ports, Optional

Name	Direction	Width	Group	Description
LOCAL_BUS_M_DATA_cmd_valid	Out	1	Local Bus Command	—
LOCAL_BUS_M_DATA_cmd_ready	In	1		—
LOCAL_BUS_M_DATA_cmd_payload_wr	Out	1		—
LOCAL_BUS_M_DATA_cmd_payload_uncached	Out	1		Fixed at 1'b0.
LOCAL_BUS_M_DATA_cmd_payload_address	Out	32		—
LOCAL_BUS_M_DATA_cmd_payload_data	Out	32		—
LOCAL_BUS_M_DATA_cmd_payload_mask	Out	4		The width field of load or store instruction.
LOCAL_BUS_M_DATA_cmd_payload_size	Out	3		3'b101: an 8-word read burst transfer. 3'b010: a single burst transfer.
LOCAL_BUS_M_DATA_cmd_payload_last	Out	1		—
LOCAL_BUS_M_DATA_cmd_payload_exclusive	Out	1		Indicates it is an atomic transaction.
LOCAL_BUS_M_DATA_rsp_valid	In	1	Local Bus Read Response	—
LOCAL_BUS_M_DATA_rsp_payload_last	In	1		—
LOCAL_BUS_M_DATA_rsp_payload_data[31:0]	In	32		—
LOCAL_BUS_M_DATA_rsp_payload_error	In	1		—
LOCAL_BUS_M_DATA_rsp_payload_exclusive	In	1		1: Indicates the exclusive store succeeds.

Name	Direction	Width	Group	Description
				0: Indicates the exclusive store fails.
LOCAL_BUS_M_DATA_inv_valid	In	1	Local Bus Invalidate Channel	—
LOCAL_BUS_M_DATA_inv_ready	Out	1		—
LOCAL_BUS_M_DATA_inv_payload_last	In	1		—
LOCAL_BUS_M_DATA_inv_payload_enable	In	1		—
LOCAL_BUS_M_DATA_inv_payload_address	In	32		—
LOCAL_BUS_M_DATA_ack_valid	Out	1	Local Bus Inv-Ack Channel	—
LOCAL_BUS_M_DATA_ack_ready	In	1		—
LOCAL_BUS_M_DATA_ack_payload_last	Out	1		—
LOCAL_BUS_M_DATA_ack_fragment_hit	Out	1		—

Table 2.10. Local Instruction Ports, Optional

Name	Direction	Width	Group	Description
LOCAL_BUS_M_INSTR_cmd_valid	Out	1	Local Bus Command	—
LOCAL_BUS_M_INSTR_cmd_ready	In	1		—
LOCAL_BUS_M_INSTR_cmd_payload_wr	Out	1		Fixed at 1’b0.
LOCAL_BUS_M_INSTR_ cmd_payload_uncached	Out	1		Fixed at 1’b0.
LOCAL_BUS_M_INSTR_cmd_payload_address	Out	32		—
LOCAL_BUS_M_INSTR_cmd_payload_data	Out	32		Fixed at 32'b0.
LOCAL_BUS_M_INSTR_cmd_payload_mask	Out	4		Fixed at 4'b0.
LOCAL_BUS_M_INSTR_cmd_payload_size	Out	3		3’b101: an 8-word read burst transfer. 3’b010: a single burst transfer.
LOCAL_BUS_M_INSTR_cmd_payload_last	Out	1		Fixed at 4'b0.
LOCAL_BUS_M_INSTR_rsp_valid	In	1	Local Bus Read Response	—
LOCAL_BUS_M_INSTR_rsp_payload_last	In	1		—
LOCAL_BUS_M_INSTR_rsp_payload_data[31:0]	In	32		—
LOCAL_BUS_M_INSTR_rsp_payload_error	In	1		—

Table 2.11. AXI Data Ports, Fixed[1]

Name	Direction	Width	Group	Description
AXI_M_DATA_AWREADY	In	1	AXI4 Manager Write Address Channel	—
AXI_M_DATA_AWVALID	Out	1		—
AXI_M_DATA_AWADDR	Out	32		—
AXI_M_DATA_AWLEN	Out	8		—
AXI_M_DATA_AWSIZE	Out	3		—
AXI_M_DATA_AWBURST	Out	2		Not implemented.
AXI_M_DATA_AWLOCK	Out	1		Not implemented.
AXI_M_DATA_AWCACHE	Out	4		Not implemented.
AXI_M_DATA_AWPROT	Out	3		Not implemented.
AXI_M_DATA_AWQOS	Out	4		Not implemented.
AXI_M_DATA_AWREGION	Out	4		Not implemented.
AXI_M_DATA_AWID	Out	1-15		Configurable.
AXI_M_DATA_WREADY	In	1	AXI4 Manager Write Data Channel	—
AXI_M_DATA_WVALID	Out	1		—
AXI_M_DATA_WDATA	Out	32		—

Name	Direction	Width	Group	Description
AXI_M_DATA_WLAST	Out	1		Not implemented.
AXI_M_DATA_WSTRB	Out	4		—
AXI_M_DATA_BVALID	In	1	AXI4 Manager Write Response Channel	—
AXI_M_DATA_BRESP	In	2		b'00: OKAY, normal access success. b'10: SLVERR, subordinate error. b'11: DECERR, decode error.
AXI_M_DATA_BID	In	1-15		Configurable
AXI_M_DATA_BREADY	Out	1		—
AXI_M_DATA_ARVALID	Out	1	AXI4 Manager Read Address Channel	—
AXI_M_DATA_ARREADY	In	1		—
AXI_M_DATA_ARCACHE	Out	4		Not implemented.
AXI_M_DATA_ARPROT	Out	3		Not implemented.
AXI_M_DATA_ARQOS	Out	4		Not implemented.
AXI_M_DATA_ARREGION	Out	4		Not implemented.
AXI_M_DATA_ARID	Out	1-15		Configurable.
AXI_M_DATA_ARADDR	Out	32		—
AXI_M_DATA_ARLEN	Out	8		—
AXI_M_DATA_ARSIZE	Out	3		—
AXI_M_DATA_ARBURST	Out	2		Fixed at 2'b01.
AXI_M_DATA_ARLOCK	Out	1		Not implemented.
AXI_M_DATA_RID	In	1-15	AXI4 Manager Read Data Channel	Configurable.
AXI_M_DATA_RDATA	In	32		—
AXI_M_DATA_RRESP	In	2		b'00: OKAY, normal access success. b'10: SLVERR, subordinate error. b'11: DECERR, decode error.
AXI_M_DATA_RLAST	In	1		—
AXI_M_DATA_RVALID	In	1		—
AXI_M_DATA_RREADY	Out	1		—

Note:

1. Optional interfaces can be configured through the RX Module/IP Block Wizard GUI upon your need. Meanwhile, the fixed interface is necessary for the RX core and cannot be configured through the GUI.

Table 2.12. AXI Instruction Ports, Optional

Name	Direction	Width	Group	Description
AXI_M_INSTR_AWREADY	In	1	AXI4 Manager Write Address Channel	Not used.
AXI_M_INSTR_AWVALID	Out	1		Not used.
AXI_M_INSTR_AWADDR	Out	32		Not used.
AXI_M_INSTR_AWLEN	Out	8		Not used.
AXI_M_INSTR_AWSIZE	Out	3		Not used.
AXI_M_INSTR_AWBURST	Out	2		Not used.
AXI_M_INSTR_AWLOCK	Out	1		Not used.
AXI_M_INSTR_AWCACHE	Out	4		Not used.
AXI_M_INSTR_AWPROT	Out	3		Not used.
AXI_M_INSTR_AWQOS	Out	4		Not used.

Name	Direction	Width	Group	Description
AXI_M_INSTR_AWREGION	Out	4		Not used.
AXI_M_INSTR_AWID	Out	1-15		Configurable
AXI_M_INSTR_WREADY	In	1	AXI4 Manager Write Data Channel	Not used.
AXI_M_INSTR_WVALID	Out	1		Not used.
AXI_M_INSTR_WDATA	Out	32		Not used.
AXI_M_INSTR_WLAST	Out	1		Not used.
AXI_M_INSTR_WSTRB	Out	4		Not used.
AXI_M_INSTR_BVALID	In	1	AXI4 Manager Write Response Channel	Not used.
AXI_M_INSTR_BRESP	In	2		Not used.
AXI_M_INSTR_BID	In	1-15		Configurable
AXI_M_INSTR_BREADY	Out	1		Not used.
AXI_M_INSTR_ARVALID	Out	1	AXI4 Manager Read Address Channel	—
AXI_M_INSTR_ARREADY	In	1		—
AXI_M_INSTR_ARCACHE	Out	4		Not implemented.
AXI_M_INSTR_ARPROT	Out	3		Not implemented.
AXI_M_INSTR_ARQOS	Out	4		Not implemented.
AXI_M_INSTR_ARREGION	Out	4		Not implemented.
AXI_M_INSTR_ARID	Out	1-15		Configurable.
AXI_M_INSTR_ARADDR	Out	32		—
AXI_M_INSTR_ARLEN	Out	8		—
AXI_M_INSTR_ARSIZE	Out	3		Fixed at 2'b10.
AXI_M_INSTR_ARBURST	Out	2		Fixed at 2'b01.
AXI_M_INSTR_ARLOCK	Out	1		Not implemented.
AXI_M_INSTR_RID	In	1-15	AXI4 Manager Read Data Channel	Configurable
AXI_M_INSTR_RDATA	In	32		—
AXI_M_INSTR_RRESP	In	2		b’00: OKAY, normal access success. b’10: SLVERR, subordinate error. b’11: DECERR, decode error.
AXI_M_INSTR_RLAST	In	1		—
AXI_M_INSTR_RVALID	In	1		—
AXI_M_INSTR_RREADY	Out	1		—

2.3.3. CXU-LI Interface

CXU-LI is used to connect the CXU accelerator. The RX core supports two CXU-LIs. You can enable CXU-LI and configure the number of CXU-LI in the Module/IP Block Wizard GUI (Figure 2.29).

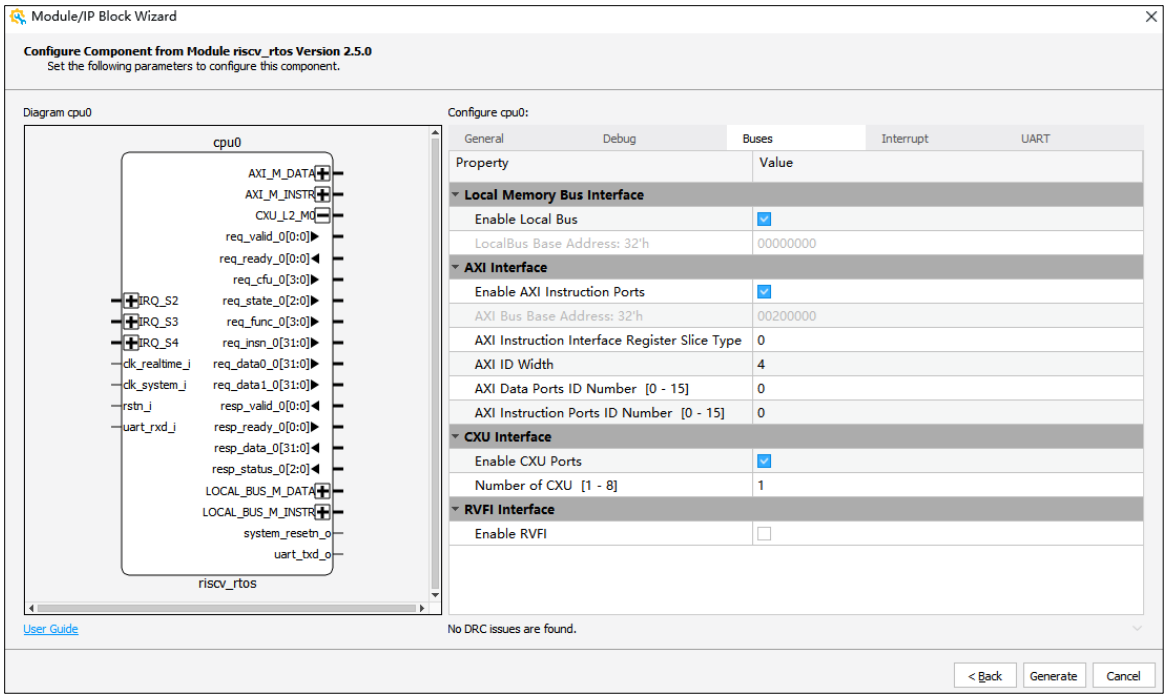


Figure 2.29. Enable CXU-LI Interface

Table 2.13. 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	4		Request CF_ID, The MSB is fixed 0.
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

2.3.4. Interrupt Interface

Table 2.14. Interrupt Ports

Name	Type	Width	Description
EXT_IRQ_Sx	In	2 ~ 31	Peripheral interrupts

2.3.5. Debug On Off Control Port

Table 2.15. Debug On Off Control Port

Name	Direction	Width	Description
debug_enable	In	1	1: debug module on. 0: debug module off.

2.3.6. Soft JTAG Interface

Table 2.16. Soft JTAG Ports

Name	Direction	Width	Description
TDI	In	1	Test data input pin
TCK	In	1	Test data output pin
TMS	In	1	Test clock pin
TDO	Out	1	Test mode select pin for controlling the TAP state machine

2.3.7. UART Ports

Table 2.17. UART Ports

Name	Direction	Width	Description
uart_txd_o	out	1	Send data pin
uart_rxd_i	In	1	Receive data pin

2.3.8. RVFI Interface

The RX core supports the instruction metadata, integer register read/write, program counter, and memory access signals of the RISC-V Formal Interface. The interface can be exposed by checking the check box in the Module/IP Block Wizard.

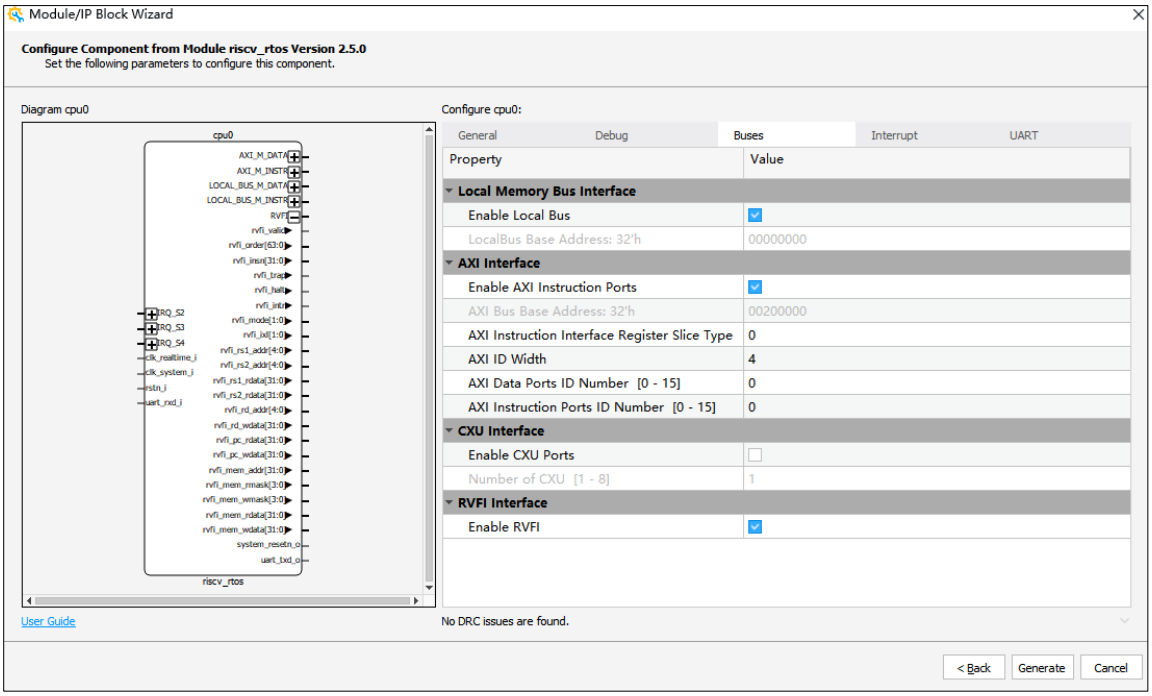


Figure 2.30. Enable RVFI Interface

The Interface consists of only output signals. As shown in Figure 2.31, when the core retires an instruction, it asserts the rvfi_valid signal and uses the signals described in Table 2.18 to output the details of the retired instruction. The 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.

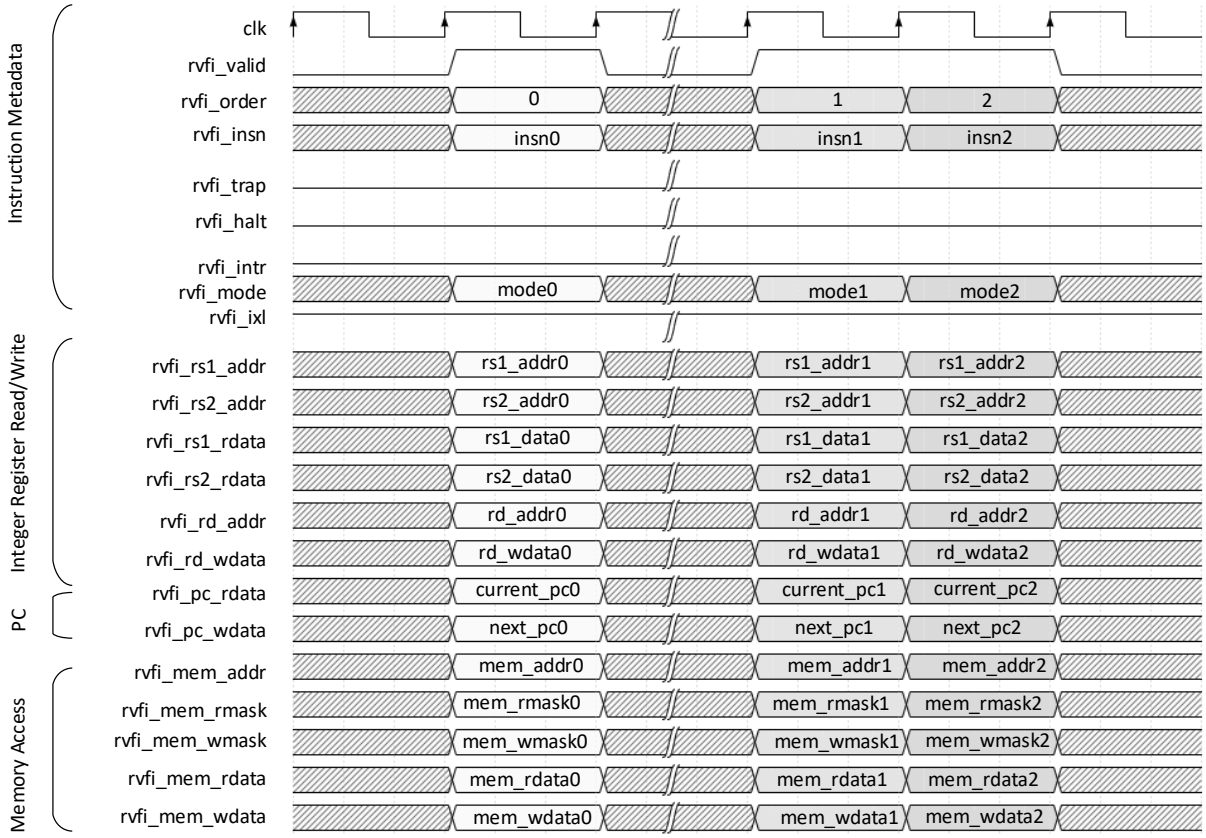


Figure 2.31. RVFI Interface

Table 2.18. 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.

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 are shown in Table 2.19 and are described in Table 2.20.

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

Table 2.19. Configurable Attributes

Attribute	Values	Default	Dependency on Other Attributes/Device Family
General			
Processor Mode	Advanced, Balanced, Lite	Balanced	—
Extension			
C Extension	Enabled	—	—
M Extension	Enabled	—	—
F Extension	Enabled	—	The F extension is enabled when the Advanced mode is selected.
	Disabled	—	The F extension is disabled when the Lite mode or Balanced mode is selected.
A Extension	Enabled	—	The A extension is enabled when the Advanced mode or Balanced mode is selected.
	Disabled	—	The A extension is disabled when the Lite mode is selected.
Reset			
Reset Vector	32'h00000000–32'hffffffff	32'h00000000	Reset Vector is editable when the Advanced mode or Balanced mode is selected.
Cache			
Cache Enable	Enabled	—	Cache Enable is enabled when the Advanced mode or Balanced mode is selected.
	Disabled		Cache Enable is disabled when the Lite mode is selected.
Instruction Cache Enable	Enabled	—	Instruction Cache Enable is enabled when the Advanced mode or Balanced mode is selected.
	Disabled		Instruction Cache is disabled when the Lite mode is selected.
Data Cache Enable	Enabled	—	Data Cache Enable is enabled when the Advanced mode or Balanced mode is selected.
	Disabled		Data Cache Enable is disabled when the Lite mode is selected.
Cache Cacheable Range High Limit	32’h00000000–32’hC0000000	32'h40000000	Valid and editable, when the Advanced mode or Balanced mode is selected.

Attribute	Values	Default	Dependency on Other Attributes/Device Family
Instruction Cache Cacheable Range Lower Limit	32'h00000000	—	Valid and editable when the Advanced mode or Balanced mode is selected.
Instruction Cache Cacheable Range Higher Limit	32'h00000000–32'hC0000000	32'h40000000	Valid and editable when the Advanced mode or Balanced mode is selected. Its value is equal to that of Cache Cacheable Range High Limit.
Data Cache Cacheable Range Lower Limit	32'h00000000	—	Valid and editable when the Advanced mode or Balanced mode is selected.
Data Cache Cacheable Range Higher Limit	32'h00000000–32'hC0000000	32'h40000000	Valid and editable when the Advanced mode or Balanced mode is selected. Its value is equal to that of Cache Cacheable Range High Limit.
Debug			
Enable Debug	Disabled, Enabled	Enabled	—
Enable Debug On Off Control Port	Enabled, Disabled	Disabled	—
JTAG			
JTAG Type	Hard, Soft	Hard	JTAG Type is selectable for Certus-N2, Lattice Avant, MachXO5-NX, CrossLink-NX, Certus-NX, CertusPro-NX devices when Enable Debug is enabled.
	Uneditable	—	JTAG Type is uneditable when Enable Debug is disabled.
Extend JTAG Channel	Enabled, Disabled	Disabled	—
JTAG Channel Selection for Certain Devices	14–24	14	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	
Buses			
Local Memory Bus Interface			
Enable Local Bus	Enabled, Disabled	Enabled	Enable Local Bus is selectable when Enable AXI Instruction Ports is enabled.
	Enabled	—	Enable Local Bus needs to be enabled when Enable AXI Instruction Ports is disabled.
AXI Interface			
Enable AXI Instruction Ports	Enabled, Disabled	Enabled	Enable AXI Instruction Ports is selectable when TCM is enabled.
	Enabled	—	Enable AXI Instruction Ports needs to be enabled when TCM is disabled.
AXI Instruction Interface Register Slice Type	0, 1, 2	0	AXI Instruction Interface Register Slice Type is selectable when Enable AXI Instruction Ports is enabled.
AXI ID Width	1–15	4	—
AXI Data Ports ID Number	0 to $2^{\text{AXI ID Width}}-1$	0	The value of AXI Data Ports ID Number is dependent on AXI ID Width.
AXI Instruction Ports ID Number	0 to $2^{\text{AXI ID Width}}-1$	0	AXI Instruction Ports ID Number is editable when Enable AXI Instruction Ports is enabled. The value of AXI Data Ports ID Number is dependent on AXI ID Width.

Attribute	Values	Default	Dependency on Other Attributes/Device Family
CXU Interface			
Enable CXU Ports	Enabled, Disabled	Disabled	—
Number of CXU	1, 2	1	Number of CXU is selectable when Enable CXU Ports is enabled.
Interrupt			
CLINT			
Enable CLINT	Enabled, Disabled	Disabled	—
PLIC Configuration			
Enable Non-maskable Interrupt	Enabled, Disabled	Disabled	—
Enable Interrupt for Supervisor Mode	Enabled, Disabled	Disabled	—
Width of Interrupt Priority Register	2, 3	3	—
Number of User Interrupt Requests	130	3	—
CDC Register			
Enable CDC Register for IRQ_SN	Enabled, Disabled	Disabled	Enable CDC Register for IRQ_SN is selectable when Number of User Interrupt Requests ≥ N-1.
	Disabled	—	Enable CDC Register for IRQ_SN is disabled when Number of User Interrupt Requests < N-1.
UART			
Local UART			
Enable UART Instance	Enabled, Disabled	Disabled	—
System Clock Frequency (MHz)	2–200	100	System Clock Frequency (MHz) is editable when Enable UART Instance is enabled.
Serial Data Width	5, 6, 7, 8	8	Serial Data Width is selectable when Enable UART Instance is enabled.
Stop Bits	1, 2	1	Stop Bits is selectable when Enable UART Instance is enabled.
Enable Parity	Enabled, Disabled	Disabled	Parity Enable is selectable when Enable UART Instance is enabled.
UART Standard Baud Rate	2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200	115200	UART Standard Baud Rate is selectable when Enable UART Instance is enabled.
Enable UART SIM	Disable, Enable	Disabled	—

Table 2.20. Attributes Description

Attribute	Description
General	
Processor Mode	Specifies the processor mode. Advanced – selects the Advanced mode. Balanced – selects the Balanced mode. Lite – selects the Lite mode.
Extension	
C Extension	Shows the support for the C extension.
M Extension	Shows the support for the M extension.
F Extension	Shows the support for the F extension.
A Extension	Shows the support for the A extension.

Attribute	Description
Reset	
Reset Vector	Reset vector initial value.
Cache	
Cache Enable	Shows the support for caches.
Instruction Cache Enable	Shows the support for the instruction cache.
Data Cache Enable	Shows the support for the data cache.
Cache Cacheable Range High Limit	Specifies the higher limit of the cache's cacheable range.
Instruction Cache Cacheable Range Lower Limit	Shows the lower limit of the instruction cache's cacheable range.
Instruction Cache Cacheable Range Higher Limit	Shows the higher limit of the instruction cache's cacheable range.
Data Cache Cacheable Range Lower Limit	Shows the lower limit of the data cache's cacheable range.
Data Cache Cacheable Range Higher Limit	Shows the higher limit of the data Cache's cacheable range.
Debug	
Enable Debug	Enables the Debug module or not.
Enable Debug On Off Control Port	Enables the presence of the Debug On Off Control port on the generated IP. Enabled – Port is available. Disabled – Port is unavailable.
JTAG	
JTAG Type	Specifies the JTAG Type.
Extend JTAG Channel	Enables the JTAG channel's range extension.
JTAG Channel Selection for Certain Devices	Specifies the channel of RX JTAG block.
Buses	
Local Memory Bus Interface	
Enable Local Bus	Enables the presence of the local bus on the generated IP. Enabled – The bus is available. Disabled – The bus is unavailable.
AXI Interface	
Enable AXI Instruction Ports	Enables the presence of AXI instruction ports on the generated IP. Enabled – The ports are available. Disabled – The ports are unavailable.
AXI Instruction Interface Register Slice Type	Type of AXI instruction ports channel register slice 0 – Bypass register slice 1 – Simple buffer 2 – Skid buffer
AXI ID Width	Specifies the AXI ID signals width.
AXI Data Ports ID Number	Specifies the value of RX AXI data ports ID signals.
AXI Instruction Ports ID Number	Specifies the value of RX AXI instruction ports ID signals.
CXU Interface	
Enable CXU Ports	Enables the presence of CXU ports on the generated IP. Enabled – The ports are available. Disabled – The ports are unavailable.
Number of CXU	Specifies the Number of CXU Ports.

Attribute	Description
Interrupt	
PLIC Configuration	
Enable Non-maskable Interrupt	Enables the presence of Non-maskable interrupt signal on the generated IP. Enabled – signal is available. Disabled – signal is unavailable.
Enable Interrupt for Supervisor Mode	Enables interrupt for the Supervisor mode. If not enabled, all external interrupts go to the Machine mode only.
Width of Interrupt Priority Register	Specifies the data width of PLIC priority register. The default is 3-bit. There are eight priority levels in total.
Number of User Interrupt Requests	Specifies the supported number of Interrupt for peripherals.
CDC Register	
Enable CDC Register for IRQ_SN	Enables the presence of 2-stage synchronizer on enabled IRQ_S interface. Enabled – The ports are available. Disabled – The ports are unavailable.
UART	
Local UART	
Enable UART Instance	Enables the local UART inside the RX. Enabled – The UART module inside RX is available. Disabled – The UART module inside RX is unavailable.
System Clock Frequency (MHz)	Specifies the target frequency of the system clock. This is used for baud rate calculation.
Serial Data Width	Specifies the default data bit width of UART transactions.
Stop Bits	Specifies the default number of stop bits to be transmitted and received.
Enable Parity	Specifies the absence/presence of parity.
UART Standard Baud Rate	Selects between Standard Baud Rate and Custom Baud Rate for the reset value of the divisor latch register. The selected baud rate is used to set the reset value of divisor latch register as follows: {DLR_MSB, DLR_LSB} = System Clock Frequency (MHz) x 1000000/Selected Baud Rate.
Enable UART SIM	Enables the function of printing strings in Questa and ModelSim transcript. Enabled – Strings are printed inside simulation tool transcript. uart_txd_o port drives uart_txd_o to output the characters signal. Disabled – The RX core does not drive uart_txd_o to output the characters signal. Strings are not printed inside simulation tool transcript. **Note:** When enabling UART SIM, the IP/Module Block Wizard shows the following error message. This option disables UART for hardware.

2.5. Memory Map

To achieve better overall performance, this IP separates the whole 4 GB memory range into several sections with some usage convention (Table 2.21 and Table 2.22).

Table 2.21. Advanced and Balanced Core SoC Memory Map

Base Address	Range	End Address	Description
Region #0 (0x0000_0000–0x0FFF_FFFF) – RISC-V RX IP			
0x0000_0000	2 MB	0x001F_FFFF	TCM, when TCM is enabled. User memory extension, when TCM is disabled.
0x0020_0000	254 MB	0x0FFF_FFFF	User cacheable memory extension, when Address < Cache Cacheable Range High Limit. User uncacheable peripheral extension, when Address >= Cache Cacheable Range High Limit.
Region #1–Region #11 (0x1000_0000–0xBFFF_FFFF) – RISC-V RX IP			
0x1000_0000	2816 MB	0x1FFF_FFFF	User cacheable memory extension, when Address < Cache Cacheable Range High Limit.

Base Address	Range	End Address	Description
			User uncacheable peripheral extension, when Address >= Cache Cacheable Range High Limit.
Region #11 - Region #14 (0xC000_0000 – 0xEFFF_FFFF) – RISC-V RX IP			
—[1]	—[1]	—[1]	User uncacheable peripheral extension.
Region #15 (0xF000_0000 – 0xFFFF_FFFF) – RISC-V RX IP			
0xF000_0000	1 KB	0xF000_03FF	Local UART, when UART_EN is asserted. Otherwise, reserved.
0xF000_0400	32767 KB	0xF1FF_FFFF	Reserved.
0xF200_0000	1024 KB	0xF20F_FFFF	CLINT and Watchdog Timer.
0xF210_0000	NA	0xFBFF_FFFF	Reserved.
0xFC00_0000	4096 KB	0xFC3F_FFFF	PLIC.
0xFC40_0000	NA	0xFFFF_FFFF	Reserved.

Note:

1. The actual valid base address/range/end address is determined by the user SoC design.

Table 2.22. Light Core SoC Memory Map

Base Address	Range	End Address	Description
Region #0 (0x0000_0000 – 0x0FFF_FFFF) – RISC-V RX IP			
0x0000_0000	2MB	0x001F_FFFF	TCM, when TCM is enabled. User memory extension, when TCM is disabled.
0x0020_0000	254MB	0x0FFF_FFFF	User extension
Region #1 - Region #14 (0x1000_0000 – 0xEFFF_FFFF) – RISC-V RX IP			
—[1]	—[1]	—[1]	User extension
Region #15 (0xF000_0000 – 0xFFFF_FFFF) – RISC-V RX IP			
0xF000_0000	1 KB	0xF000_03FF	Local UART, when UART_EN is asserted. Otherwise, reserved.
0xF000_0400	32767 KB	0xF1FF_FFFF	Reserved.
0xF200_0000	1024 KB	0xF20F_FFFF	CLINT and Watchdog Timer.
0xF210_0000	NA	0xFBFF_FFFF	Reserved.
0xFC00_0000	4096 KB	0xFC3F_FFFF	PLIC.
0xFC40_0000	NA	0xFFFF_FFFF	Reserved.

Note:

1. The actual valid base address/range/end address is determined by the user SoC design.

The total 4 GB memory space is divided into 16 256 MB regions to ease potential future PMP settings.

For the Advanced and Balanced modes, the SoC memory map assignment is relevant to Cache Cacheable Range High Limit. The lower limit of the processor's cacheable range is fixed at 0x0000_0000. The higher limit of the processor's cacheable range is configurable from 0x0000_0000 to 0xBFFF_FFFF.

The first 2 MB of region #0, from 0x0000_0000 to 0x0001_FFFF, are reserved for TCM. When the address is below Cache Cacheable Range High Limit, the remaining spaces of region #0 and region #1 to region #11 are for user external memory extension, either for on-chip EBR-based memory or off-chip memory like flash and SDRAM. When the address is beyond Cache Cacheable Range High Limit, the remaining spaces of region #0 and region #1 to region #11 are for user uncacheable peripheral extension.

From region #11 to region #14, the address spaces are for user uncacheable peripheral extension.

For the Lite mode, the processor does not have a cache. The first 2 MB of region #0, from 0x0000_0000 to 0x0001_FFFF, are reserved for TCM. The remaining spaces of region #0 and region #1 to region #14 are for user extension.

For all three modes, region #15 is reserved for the RISC-V RX IP. Local UART, CLINT, Watchdog Timer, and PLIC are assigned to this region.

3. RISC-V RX CPU IP Generation

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

To generate the RX IP Core module:

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

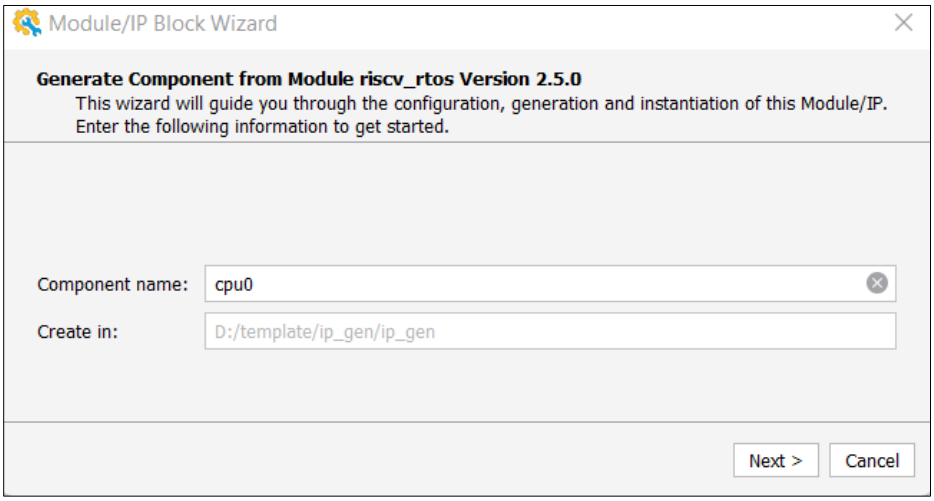


Figure 3.1. Entering Component Name

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

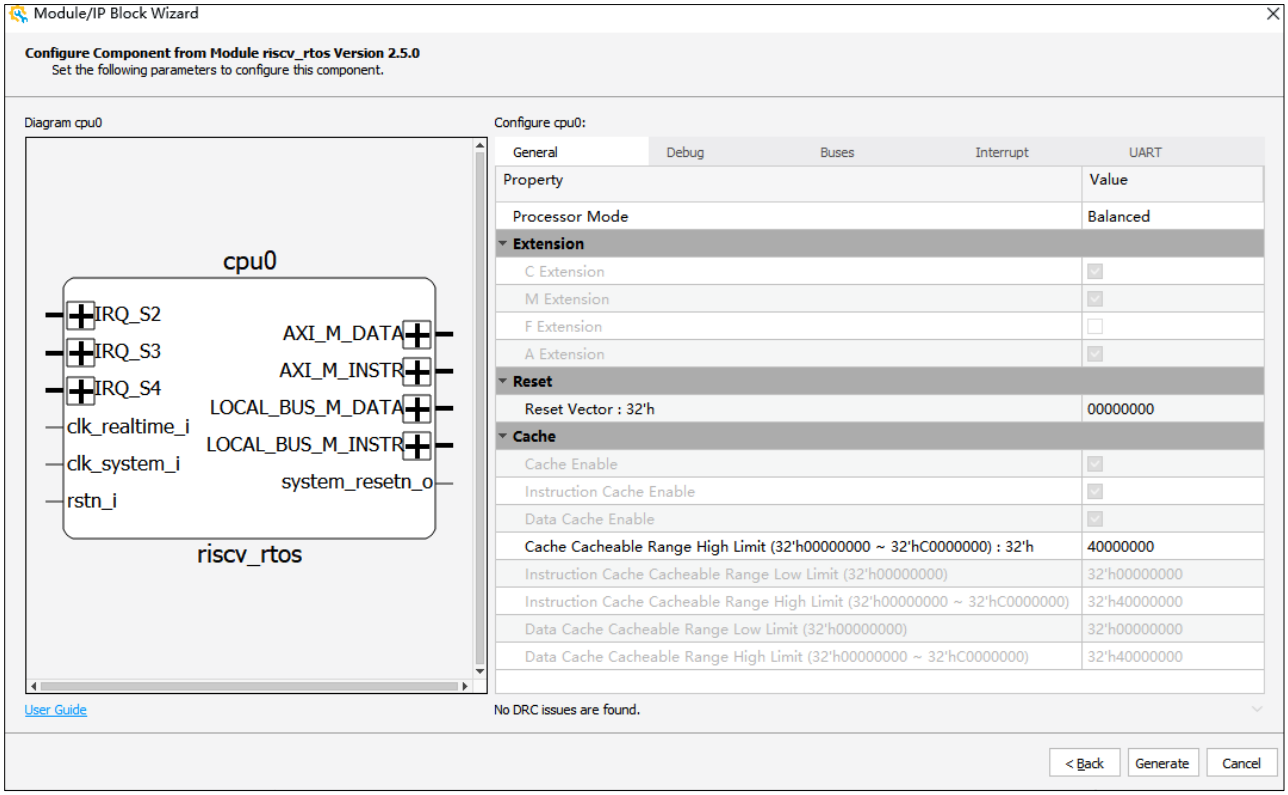


Figure 3.2. Configuring Parameters

4. Verify the information, as shown in Figure 3.3. Click **Finish**.

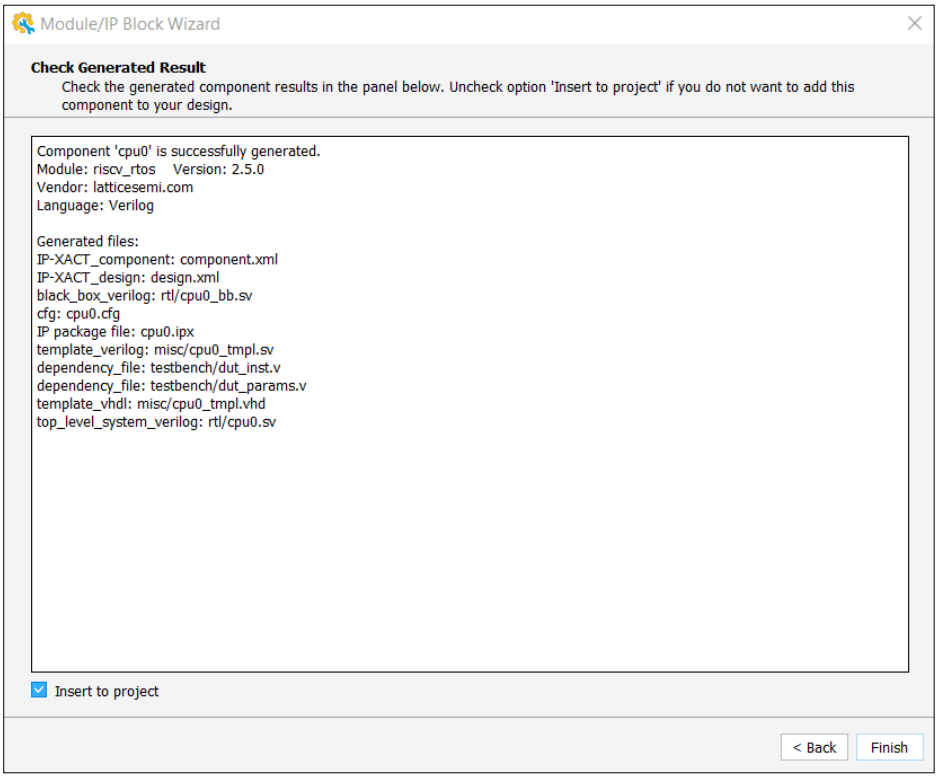


Figure 3.3. Verifying Results

5. Confirm or modify the module instance name, as shown in Figure 3.4. Click **OK**.

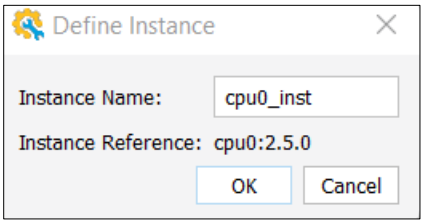


Figure 3.4. Specifying Instance Name

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

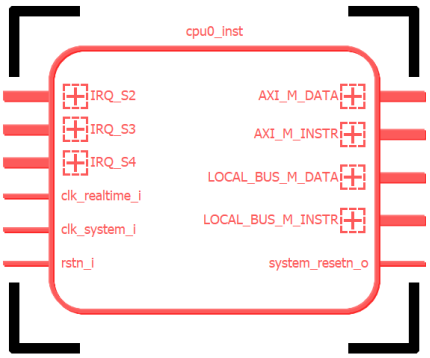


Figure 3.5. Generated Instance

Appendix A. Resource Utilization

Table A.1. Resource Utilization in CertusPro-NX Device

Configuration	LUTs	Registers	sysMEM EBRs
Processor Advanced core	9547	4972	21
Processor Balanced core	5770	2783	18
Processor Lite core	3876	1869	2
Processor Advanced core + PLIC + CLINT + CXU-LI + Debug	10476	5721	21
Processor Balanced core + PLIC + CLINT + CXU-LI + Debug	6718	3465	18
Processor Lite core + PLIC + CLINT + CXU-LI + Debug	4831	2685	2

Note: Resource utilization characteristics are generated using Lattice Radiant 2023.2 software.

Table A.2. Resource Utilization in Lattice Avant Device

Configuration	LUTs	Registers	sysMEM EBRs
Processor Advanced core	9852	5034	15
Processor Balanced core	5914	2816	15
Processor Lite core	4197	1843	0
Processor Advanced core + PLIC + CLINT + CXU-LI + Debug	10716	5680	15
Processor Balanced core + PLIC + CLINT + CXU-LI + Debug	6767	3461	15
Processor Lite core + PLIC + CLINT + CXU-LI + Debug	5035	2630	0

Note: Resource utilization characteristics are generated using Lattice Radiant 2023.2 software.

Appendix B. Debug with Soft JTAG

To debug with Soft JTAG:

1. In Lattice Propel Builder software, select **Soft JTAG** in the IP block Wizard GUI (Figure 2.7) when generating the IP.
2. After the IP is generated, right-click on the **JTAG** port and select **Export** (Figure B.1).

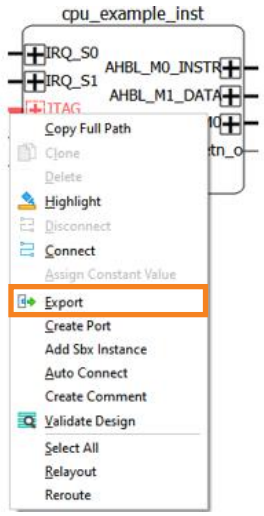
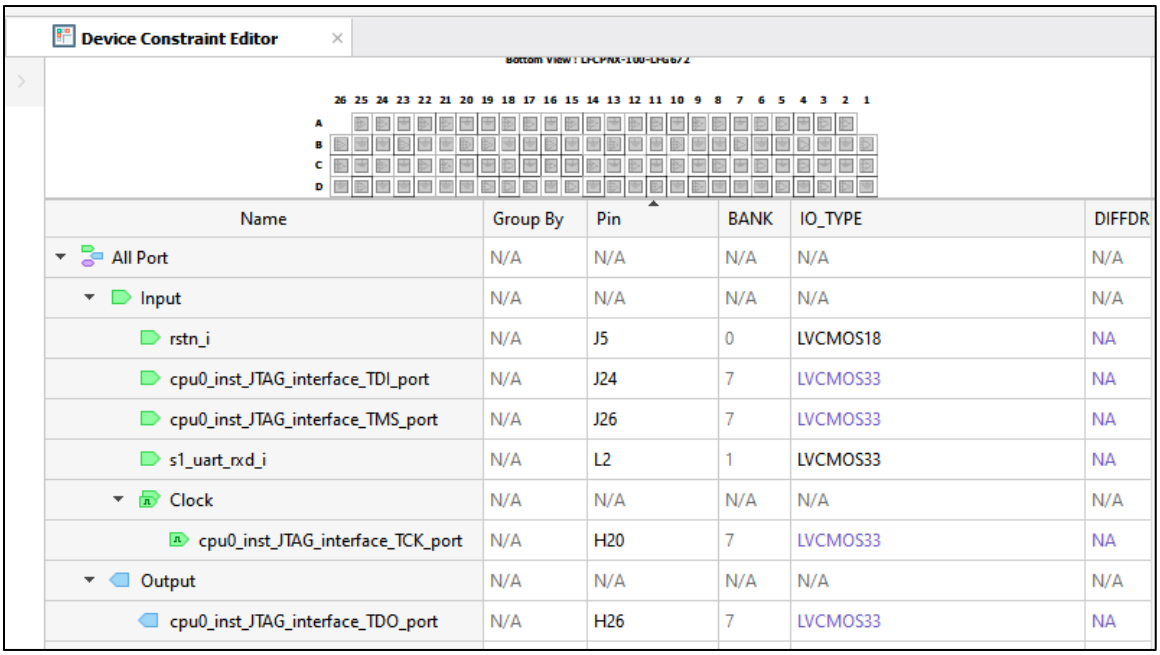


Figure B.1. Exporting Pins

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).



Name	Group By	Pin	BANK	IO_TYPE	DIFFDR
All Port	N/A	N/A	N/A	N/A	N/A
Input	N/A	N/A	N/A	N/A	N/A
rstn_i	N/A	J5	0	LVCMOS18	NA
cpu0_inst_JTAG_interface_TDI_port	N/A	J24	7	LVCMOS33	NA
cpu0_inst_JTAG_interface_TMS_port	N/A	J26	7	LVCMOS33	NA
s1_uart_rxd_i	N/A	L2	1	LVCMOS33	NA
Clock	N/A	N/A	N/A	N/A	N/A
cpu0_inst_JTAG_interface_TCK_port	N/A	H20	7	LVCMOS33	NA
Output	N/A	N/A	N/A	N/A	N/A
cpu0_inst_JTAG_interface_TDO_port	N/A	H26	7	LVCMOS33	NA

Figure B.2. Assigning Pins

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" to the Value of **Command Line Options** if TCK connects to normal I/O (Figure B.3).

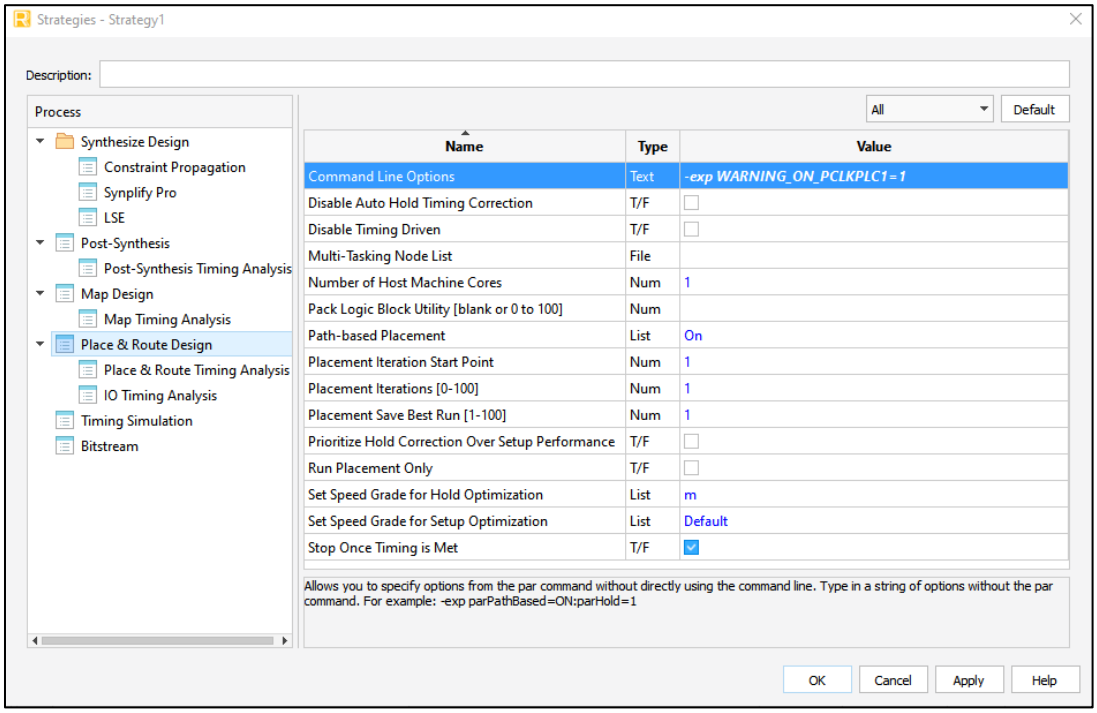


Figure B.3. Setting Environment Variables

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 Propel SDK 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.

References

- RISC-V Composable Custom Extensions Specification (Draft)
- RISC-V Instruction Set Manual Volume I: Unprivileged ISA (20191213)
- RISC-V Instruction Set Manual Volume II: Privileged Architecture (20211203)
- RISC-V Privileged Specification Version 1.12
- RISC-V Platform Specification Version 0.2
- RISC-V Platform-Level Interrupt Controller Specification Version 1.0
- SiFive Interrupt Cookbook v1.2
- RISC-V Watchdog Timer Specification Version 1.0-draft-0.5
- AMBA 3 AHB-Lite Protocol v1.0
- AMBA AXI and ACE Protocol Specification vF.b
- Local Bus Specification
- Lattice Propel Builder 2024.1 User Guide (FPGA-UG-02219)
- RISC-V Formal Interface Specification

For more information, refer to:

- Lattice Propel Design Environment 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
- Lattice Insights for Lattice Semiconductor Training Series and Learning Plans

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.

Revision History

Revision 1.0, IP v2.5.0, November 2024

Section	Change Summary
All	Production release.

www.latticesemi.com