



Mutex IP – Lattice Propel Builder 2025.2

IP Version: v1.0.0

User Guide

FPGA-IPUG-02307-1.0

December 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.

Contents

Contents	3
Abbreviations in This Document.....	5
1. Introduction	6
1.1. Quick Facts	6
1.2. Features	6
1.3. Conventions	6
1.3.1. Nomenclature.....	6
1.3.2. Signal Names	6
1.4. Licensing and Ordering Information	6
2. Functional Descriptions	7
2.1. Overview	7
2.2. Modules Description	8
2.2.1. Bus Interface	8
2.2.2. Registers	8
2.3. Program Process	8
2.4. Signal Description.....	8
2.4.1. Clock and Reset	8
2.4.2. AXI-Lite Interface.....	8
2.4.3. AHB-Lite Interface	9
2.5. Attribute Summary.....	9
3. Mutex IP Generation	11
4. Example Design.....	13
4.1. Supported Configuration.....	13
4.2. Overview of the Example Design and Features	13
4.3. Generating the Example Design.....	14
4.4. Running the Example Design.....	22
4.4.1. Setting Up the UART Terminal.....	22
4.4.2. Programming the FPGA Bitstream	22
4.4.3. Verifying Results.....	25
Appendix A. Resource Utilization	26
References.....	27
Technical Support Assistance	28
Revision History	29

Figures

Figure 2.1. Mutex IP Diagram	7
Figure 3.1. Entering Component Name	11
Figure 3.2. Configuring Parameters	11
Figure 3.3. Verifying Results	12
Figure 3.4. Specifying Instance Name	12
Figure 3.5. Generated Instance	12
Figure 4.1. RISC-V MC Multi-Processor Project with Mutex IP Block Diagram	14
Figure 4.2. Lattice Propel Builder – Create New SoC Design	15
Figure 4.3. Create Project – Design Information	15
Figure 4.4. Select Template – RISC-V MC Multi-Processor Project	16
Figure 4.5. Select Target Device – Avant-E Evaluation (ES1)	16
Figure 4.6. Lattice Propel Builder – Generate SoC Design	17
Figure 4.7. Lattice Propel SDK – Select Workspace and Launch	17
Figure 4.8. Lattice Propel SDK – Create New C/C++ Project	18
Figure 4.9. Load System and BSP – Select cpu0_inst	19
Figure 4.10. Lattice Propel SDK – Build All Projects	20
Figure 4.11. File Explorer – .mem Files for Each CPU Project	20
Figure 4.12. Lattice Propel Builder – Initialize System Memory with .mem Files	21
Figure 4.13. Lattice Propel Builder – Regenerate Design After Memory Initialization	21
Figure 4.14. Launch Lattice Radiant – Compile Design	22
Figure 4.15. Launching Radiant Programmer	23
Figure 4.16. Lattice Radiant Programmer – Device and Bitstream Configuration	23
Figure 4.17. Cable Setup – Detecting Cable	24
Figure 4.18. Select Cable – FTUSB-0 Option	24
Figure 4.19. Output Console – Programming the FPGA Bitstream	25
Figure 4.20. UART Terminal – Expected Output Messages	25

Tables

Table 1.1. Mutex IP Quick Facts	6
Table 2.1. Register Memory Map	8
Table 2.2. Clock and Reset Ports	8
Table 2.3. AXI-Lite Subordinate Ports	8
Table 2.4. AHB-Lite Subordinate Ports	9
Table 2.5. Attributes Summary	10
Table 2.6. Attributes Description	10
Table 4.1. Mutex IP Configuration Used in the Example Design	13
Table 4.2. Load System and BSP – Select Remaining CPUs	20
Table 4.3. Initialize System Memory with .mem Files	21
Table A.1. Resource Utilization in CertusPro-NX-100 Device	26
Table A.2. Resource Utilization in LAV-AT-E70 Device	26

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AXI-Lite	Advanced eXtensible Interface – Lite
AHB-Lite	Advanced High-Performance Bus – Lite
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphic User Interface
HDL	Hardware Description Language
IOPMP	I/O Physical Memory Protection
IP	Intellectual Property
IRQ	Interrupt Request
LUT	Look Up Table
MC	Micro-Controller (RISC-V for Micro-Controller applications)
PIC	Programmable Interrupt Controller
RISC-V	Reduced Instruction Set Computer-V (Five)
SoC	System-on-Chip
UART	Universal Asynchronous Receiver/Transmitter
WARL	Write Any Read Legal

1. Introduction

The Lattice Semiconductor Mutex IP is used in multi-processor environment to solve the competition of the shared resources between different processors. The Mutex provides a configurable number of registers for processors to claim they gain exclusive access to particular resources, like shared memory space or shared peripherals.

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

This document includes the description of a new example design that incorporates the Mutex IP and can be generated using the RISC-V MC Multi-Processor Project template available in Lattice Propel 2025.2. Refer to the [Example Design](#) section for details.

1.1. Quick Facts

[Table 1.1](#) presents a summary of the Mutex IP.

Table 1.1. Mutex IP Quick Facts

IP Requirements	Supported Devices	Certus-N2, Lattice Avant, MachXO5-NX, CrossLinkU-NX, CrossLink-NX, CertusPro-NX, and Certus-NX
Resource Utilization	Supported User Interfaces	AXI-Lite Interface and AHB-Lite Interface
	Resources	See Table A.1 and Table A.2 .
Design Tool Support	Lattice Implementation	IP v1.0.0 – Lattice Propel Builder 2025.2, Lattice Radiant™ Software 2025.2
	Simulation	For a list of supported simulators, see the Lattice Radiant software user guide.

1.2. Features

The Mutex IP has the following features:

- AXI-Lite interface and AHB-Lite interface
- Configurable number of mutex
- Configurable interface numbers
- Configurable CPU ID Width to clarify the owner of mutex
- Selectable hardware identification support

1.3. Conventions

1.3.1. Nomenclature

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

1.3.2. Signal Names

Signal Names that end with:

- `_n` are active low.
- `_i` are input signals.
- `_o` are output signals.
- `_io` are bi-directional input/output signals.

1.4. Licensing and Ordering Information

The Mutex 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

In a multi-processor system, there can be multiple processors sharing the same resources, such as memory space or peripherals. The Mutex IP provides several mutex registers for processors to claim exclusive access to one or more resources. A typical connection of the Mutex IP is shown in [Figure 2.1](#).

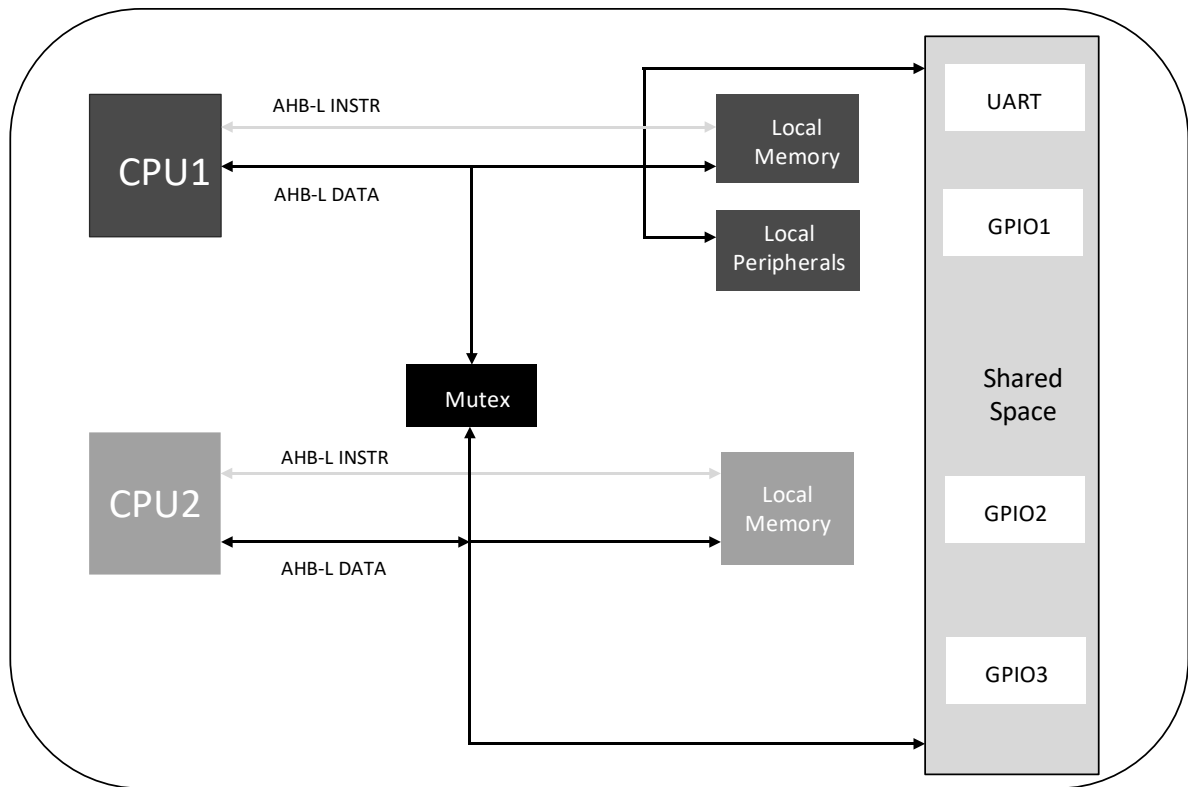


Figure 2.1. Mutex IP Diagram

2.2. Modules Description

2.2.1. Bus Interface

The Mutex IP supports two to eight bus interfaces for a multi-processor system. The interface type can be set as the AXI-Lite interface or the AHB-Lite interface.

2.2.2. Registers

The Mutex IP provides configurable numbers of mutex register. Each mutex register contains a lock bit and a field to claim the identification of the processor. The mutex registers provide a mechanism for mutual exclusion.

Table 2.1. Register Memory Map

Address	Register Name	Access	Description	Fields
0x100 * i	Mutex[i]	WARL	Mutex Register, i = 0 to Mutex Number	<p>[ID_WIDTH:1] Identification: CPU ID to clarify the owner of the mutex.</p> <p>[0] Lock: Indicates whether the mutex is locked.</p> <ul style="list-style-type: none"> 0 = Mutex is free. 1 = Mutex is locked.

2.3. Program Process

To lock a mutex register:

1. Write the Identification field of the target mutex register and set the Lock bit to one.
2. Read back the mutex register to verify the ownership of the register and the status of the lock bit. If the identification field matches the current processor, then the mutex register is owned by the processor.

To free a mutex register:

1. Write the identification field of the mutex register and set the lock bit to zero.
2. Read back the mutex register to verify that the ownership no longer belongs to the current processor.

If Port Protection is checked, when the processor is trying to lock one mutex register, its port ID is stored in another register. The same port ID is later used to free the locked mutex register. It protects the mutex from being unlocked by other processors using the same Identification number.

2.4. Signal Description

Table 2.2 to Table 2.4 list the ports of the soft IP in different categories.

2.4.1. Clock and Reset

Table 2.2. Clock and Reset Ports

Name	Direction	Width	Description
clk	In	1	Mutex soft IP clock.
rst_n	In	1	Global reset, active low.

2.4.2. AXI-Lite Interface

Table 2.3. AXI-Lite Subordinate Ports

Name	Direction	Width	Group	Description
AWREADY_Sx	Out	1	AXI4 Mandatory Write Address Channel	—
AWVALID_Sx	In	1		—
AWADDR_Sx	In	32		—

Name	Direction	Width	Group	Description
AWPROT_Sx	In	3	AXI4 Mandatory Write Data Channel	Not implemented.
WREADY_Sx	Out	1		—
WVALID_Sx	In	1		—
WDATA_Sx	In	32		—
WSTRB_Sx	In	4		—
BVALID_Sx	Out	1	AXI4 Mandatory Write Response Channel	—
BRESP_Sx	Out	2		b'00: OKAY ¹ b'10: SLVERR ¹ b'11: DECERR ¹
BREADY_Sx	In	1		—
ARVALID_Sx	In	1	AXI4 Mandatory Read Address Channel	—
ARREADY_Sx	Out	1		—
ARADDR_Sx	In	32		—
RDATA_Sx	Out	32		—
RRESP_Sx	Out	2		b'00: OKAY ¹ b'10: SLVERR ¹ b'11: DECERR ¹
RVALID_Sx	Out	1		—
RREADY_Sx	In	1		—

Note:

1. Refer to the [AMBA AXI Protocol Specification](#) for more detailed descriptions of these responses.
2. Sx represents the subordinate number from 0 to 7.

2.4.3. AHB-Lite Interface

Table 2.4. AHB-Lite Subordinate Ports

Name	Direction	Width	Description
AHBL_Sx_HADDR	In	32	—
AHBL_Sx_HWRITE	In	1	—
AHBL_Sx_HSIZE	In	3	—
AHBL_Sx_HPROT	In	4	Not implemented.
AHBL_Sx_HTRANS	In	2	—
AHBL_Sx_HBURST	In	3	Not implemented.
AHBL_Sx_HMASTLOCK	In	1	Not implemented.
AHBL_Sx_HWDATA	In	32	—
AHBL_Sx_HRDATA	Out	32	—
AHBL_Sx_HREADY	Out	1	—
AHBL_Sx_HRESP	Out	1	—

Note:

1. Refer to the [AMBA3 AHBL Protocol Specification](#) for more detailed descriptions of these responses.
2. Sx represents the subordinate number from 0 to 7.

2.5. Attribute Summary

The configurable attributes of the Mutex IP are described in [Table 2.5](#).

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

Table 2.5. Attributes Summary

Attribute	Selectable Values	Default	Dependency on Other Attributes
Mutex Number	1 to 16	2	—
Port Number	2 to 8	2	—
Port Protection	Checked, Unchecked	Unchecked	—
CPU ID Width	$\text{clog}(\text{Port Number})$ 1 to 5	4	—
Interface Type	AXI-L, AHB-L	AXI-L	—

Note:

1. $\text{clog}(x) = \text{ceil}(\log_2(x))$. The Minimum integer is greater than \log_2 of port number.

Table 2.6. Attributes Description

Attribute	Description
Mutex Number	Total mutex register number
Port Number	Number of bus interfaces
Port Protection	Hardware protection option
CPU ID Width	The bit width of ID number
Interface Type	Bus interfaces type

3. Mutex IP Generation

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

To generate the Mutex IP module:

1. In the Lattice Propel Builder software, create a new design. Select the Mutex package.
2. Enter the component name. Click **Next**, as shown in [Figure 3.1](#).

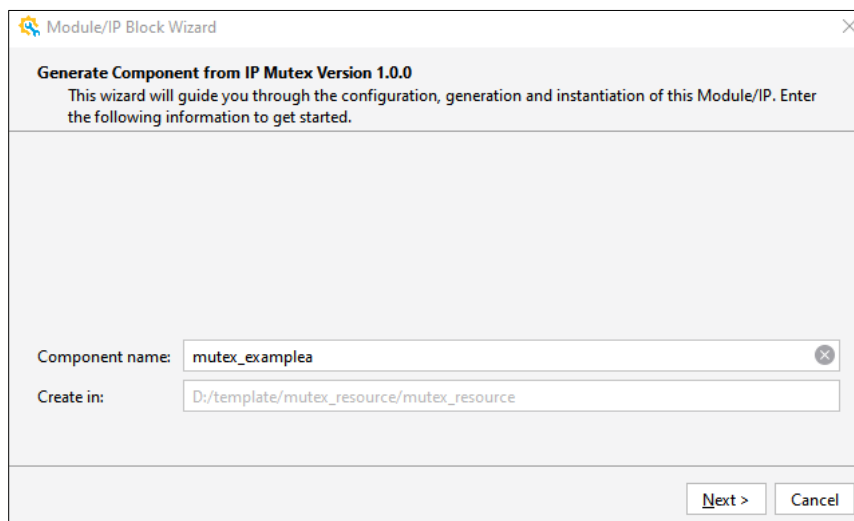


Figure 3.1. Entering Component Name

3. Configure the parameters as needed. Click **Generate** ([Figure 3.2](#)).

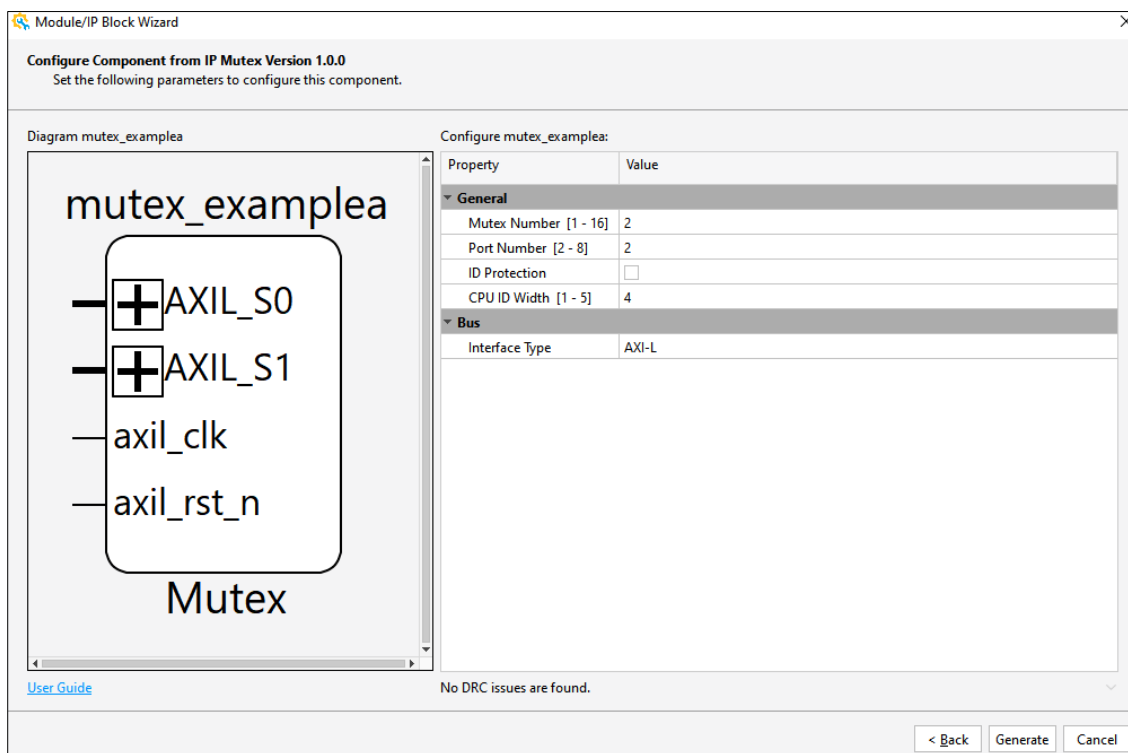


Figure 3.2. Configuring Parameters

- Verify the information. Click **Finish** (Figure 3.3).

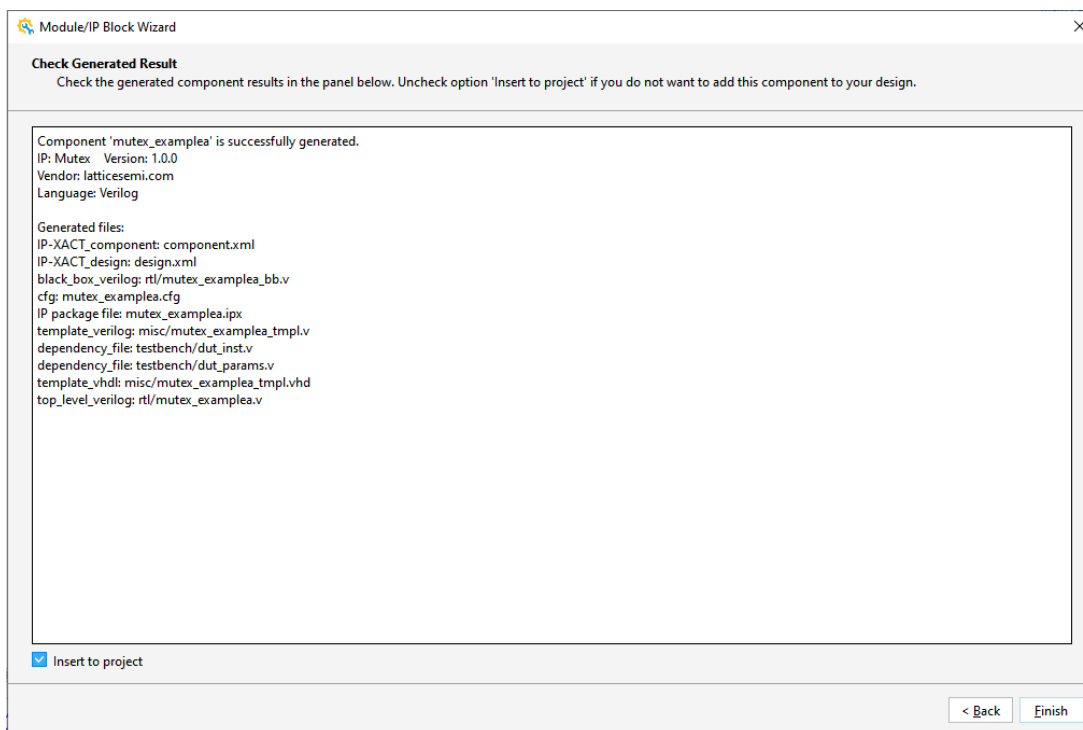


Figure 3.3. Verifying Results

- Confirm or modify the module instance name. Click **OK** (Figure 3.4).

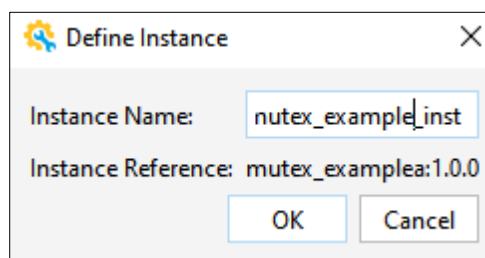


Figure 3.4. Specifying Instance Name

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

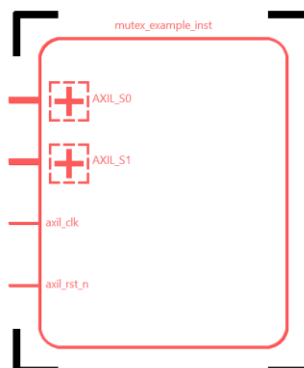


Figure 3.5. Generated Instance

4. Example Design

A system design that incorporates the Mutex IP can be generated using the RISC-V MC Multi-Processor Project template available in Lattice Propel 2025.2. This design is referred to as the Example Design throughout this guide. The Example Design is documented in the accompanying README file and validated through a C code test sequence. Additional notes and explanations are provided as comments within the test code to help you understand the implementation and behavior of the Mutex IP.

You can find the README.txt file at the following location:

<Propel Installation Path>/templates/MULTI_PROCESSOR_Template01/Readme.txt

This example design enables you to compile and test the Mutex IP functionality on the following Lattice evaluation board:

Avant-E Evaluation Board

4.1. Supported Configuration

In the example design, the following IP configuration is applied to the Mutex IP.

Table 4.1. Mutex IP Configuration Used in the Example Design

Mutex IP GUI Parameter	Mutex IP Configuration
FIFO Settings	
Mutex Number [1–16]	2
Port Number [2–8]	6
Port Protection	Unchecked
CPU ID Width [3–5]	4
Bus	
Interface Type	AHB-Lite

4.2. Overview of the Example Design and Features

The example design discussed in this section is generated using the RISC-V MC Multi-Processor Project template in the Lattice Propel Development Suite. The project generated includes the following components:

- Processor
 - CPU0 (Main) - RISC-V MC with PIC and timer
 - CPU1-5 – RISC-V MC with PIC
- AHBL to APB Bridge
- AHBL Interconnect
- APB Interconnect
- GPIO
- Mailbox
- Mutex
- IOPMP
- UART
- System Memory

This example design demonstrates the functionality of the Mutex IP in a multi-core system.

The system comprises six CPUs, from CPU0 to CPU5. CPU0 acts as the system controller, responsible for distributing tasks to CPU1–CPU5. These worker CPUs receive their assigned tasks through the Mailbox, execute the computations independently, and then send their results and status back to CPU0 using the same Mailbox mechanism.

For status output, all CPUs share a common UART interface. To prevent concurrent access to the UART, a Mutex is employed. This ensures that only one CPU can transmit data at a time, while others wait until the Mutex is released, maintaining data integrity and avoiding conflicts.

Figure 4.1 shows the block diagram of the RISC-V MC Multi-Processor project with the Mutex IP.

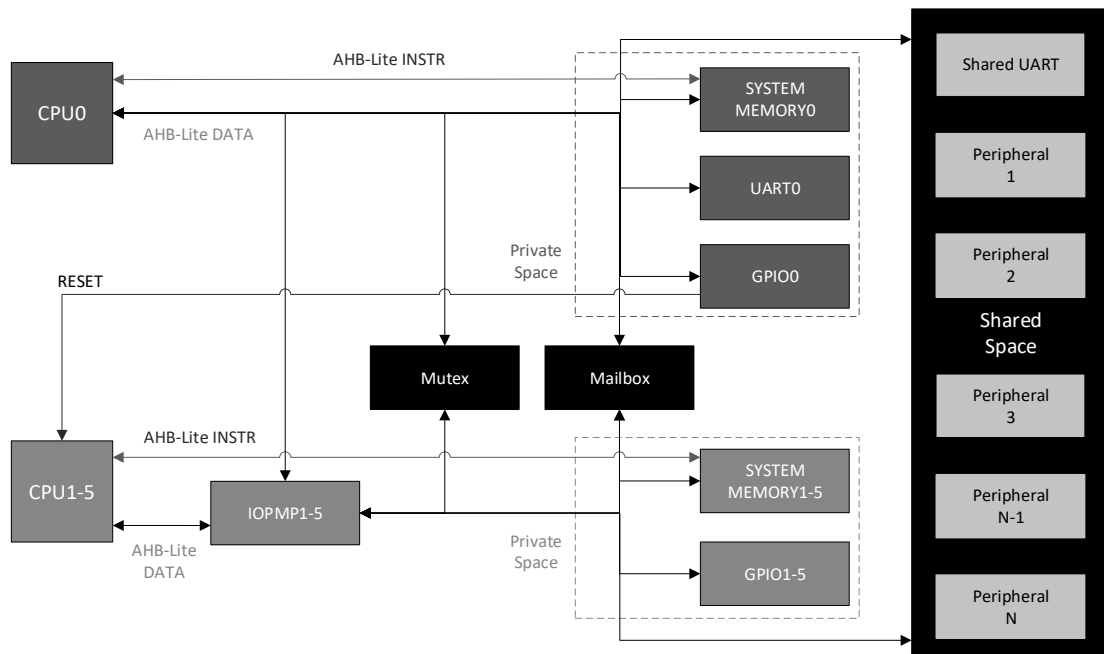


Figure 4.1. RISC-V MC Multi-Processor Project with Mutex IP Block Diagram

Note: The block diagram does not cover all components of the example design and serves only as a guideline.

4.3. Generating the Example Design

The Lattice Propel Builder and Lattice Propel SDK are used to create an embedded system that includes an SoC design with an embedded processor and system software. This section outlines the procedure for generating an example design that incorporates the Mutex IP. The hardware design is created using Lattice Propel Builder, while the software workspace is set up and managed within Lattice Propel SDK to develop, build, and deploy the corresponding firmware for each processor core.

To generate the example design:

1. Launch Lattice Propel Builder.

Open the application, then navigate to **File > New SoC Design**, as shown in Figure 4.2.

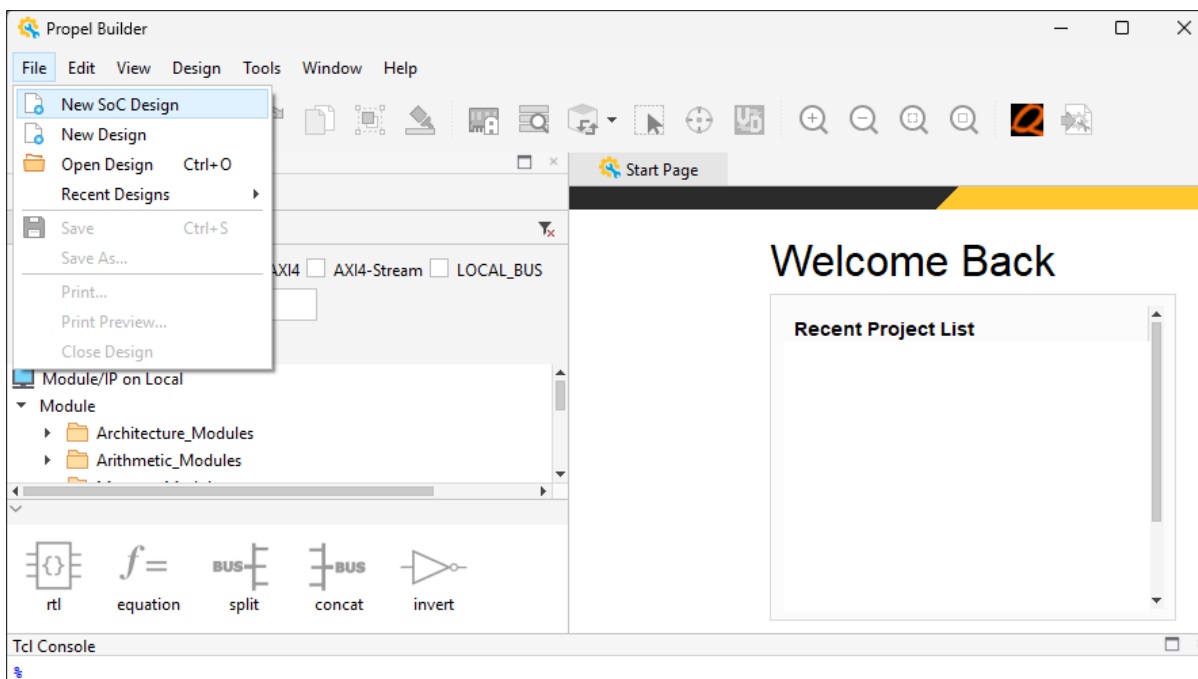


Figure 4.2. Lattice Propel Builder – Create New SoC Design

2. Enter the project Information.

In the **Create Project > Design Information** window:

- Enter a name for your project.
- Browse to select a directory for the project location.
- Click **Next** to proceed (Figure 4.3).

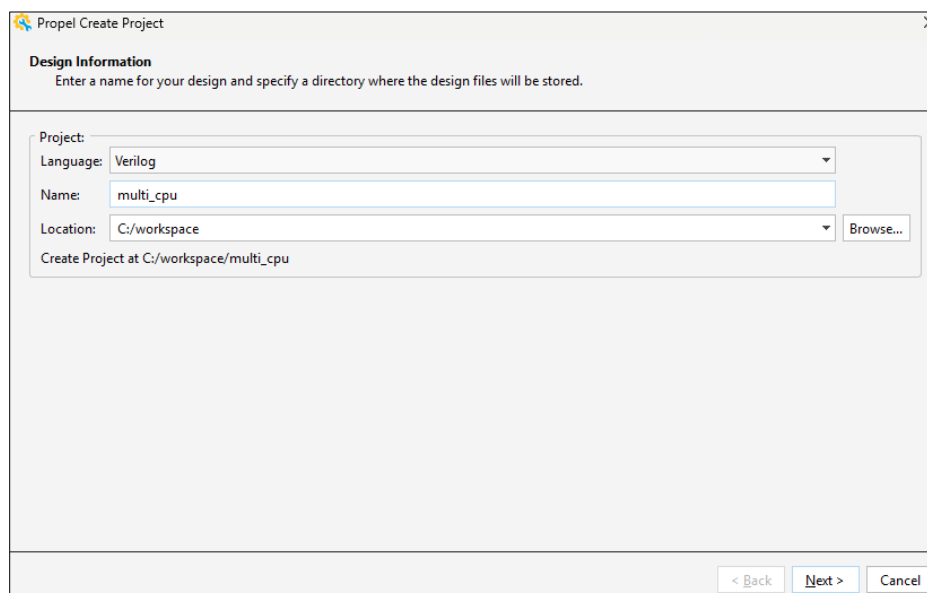


Figure 4.3. Create Project – Design Information

3. Select a Project Template.

In the **Create Project > Select Template** window:

- Choose **RISC-V MC Multi-Processor Project** from the list of available templates.
- Click **Next** to continue (Figure 4.4).

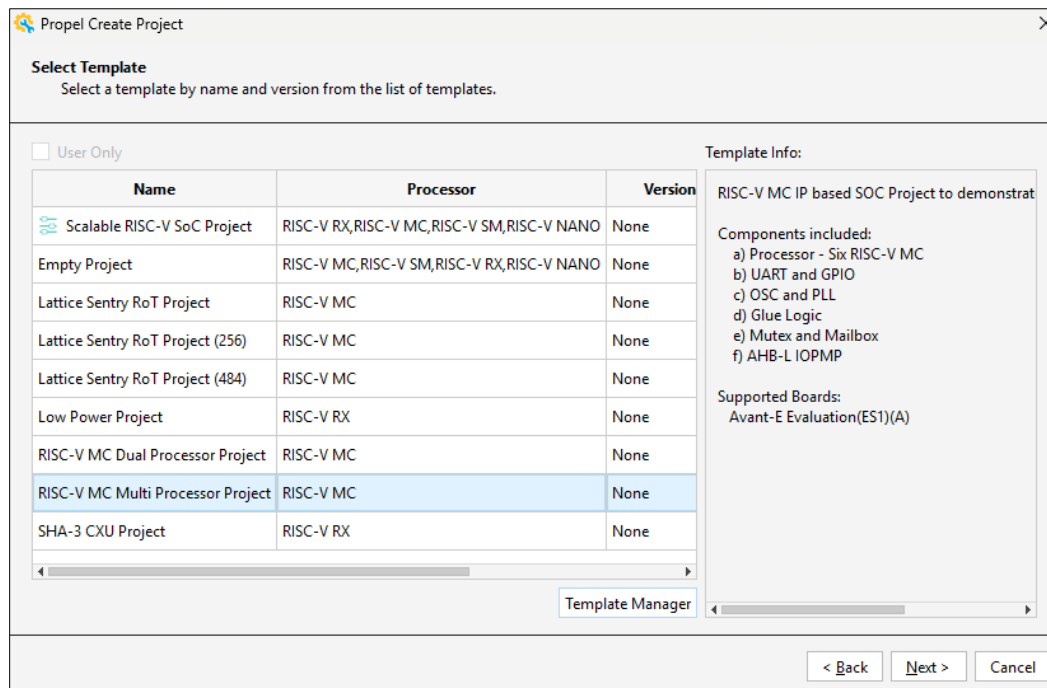


Figure 4.4. Select Template – RISC-V MC Multi-Processor Project

- Select the Target Device.
In the **Create Project > Select Device** window:
 - Select **Avant-E Evaluation (ES1)** as the target device.
 - Click **Next**, then click **Finish** to complete the project setup (Figure 4.5).

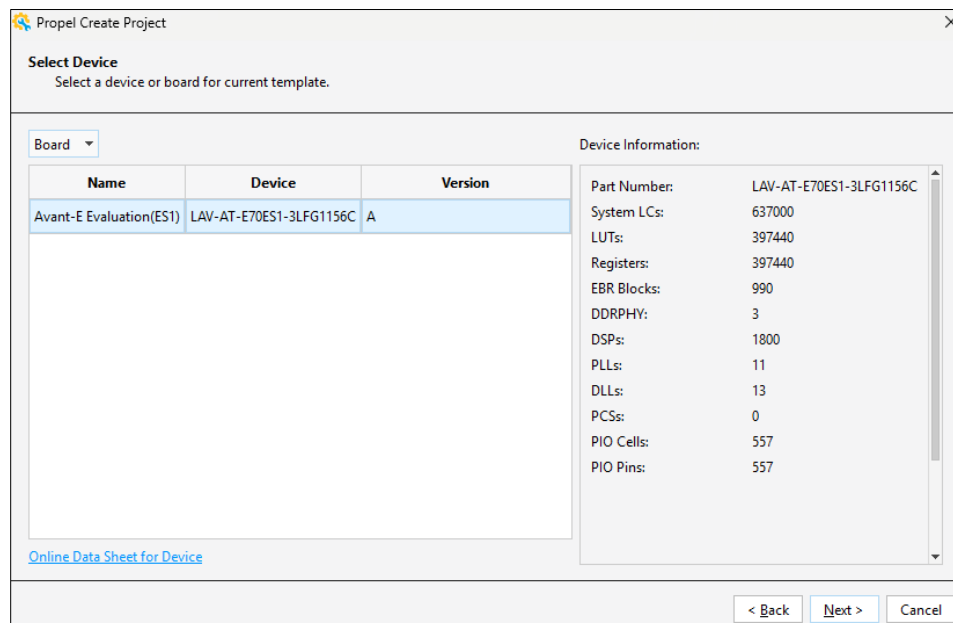


Figure 4.5. Select Target Device – Avant-E Evaluation (ES1)

5. Generate the SoC design.
 - a. Once the project is created, the **Lattice Propel Builder** main window opens.
 - b. Click the **Generate** button to build the SoC design (Figure 4.6).

Note: Warnings related to TIMER_IRQ and INT may appear. These are expected and can be safely ignored.

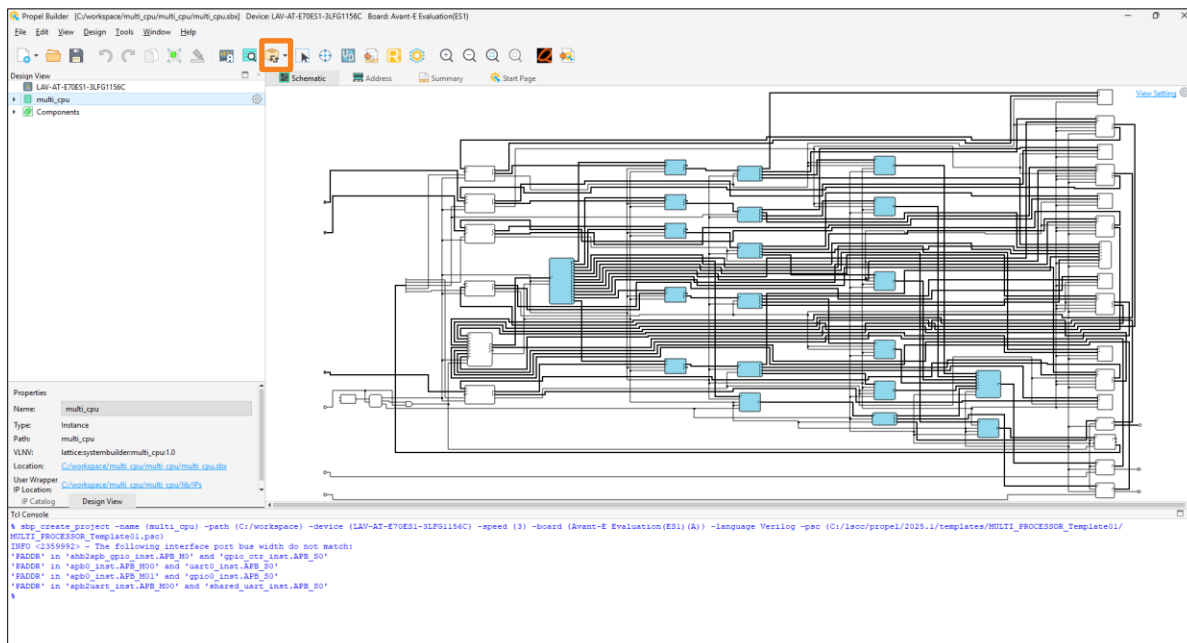


Figure 4.6. Lattice Propel Builder – Generate SoC Design

6. Launch Lattice Propel SDK.
 - a. Open Lattice Propel SDK. The Lattice Propel Launcher window appears.
 - b. Browse to select a directory for your workspace.
 - c. Click **Launch** to open the SDK environment (Figure 4.7).

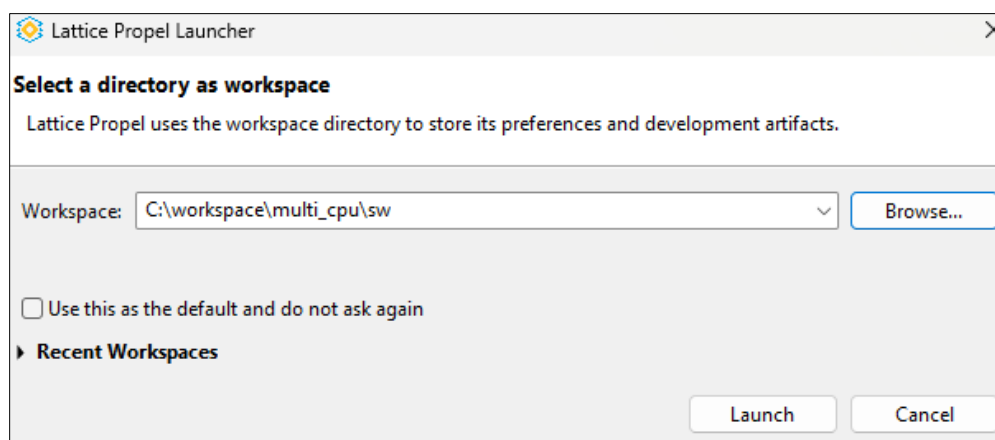


Figure 4.7. Lattice Propel SDK – Select Workspace and Launch

7. Create a New C/C++ Project.

In Lattice Propel SDK:

Navigate to **File > New > Lattice C/C++ Project** to begin creating a new software project (Figure 4.8).

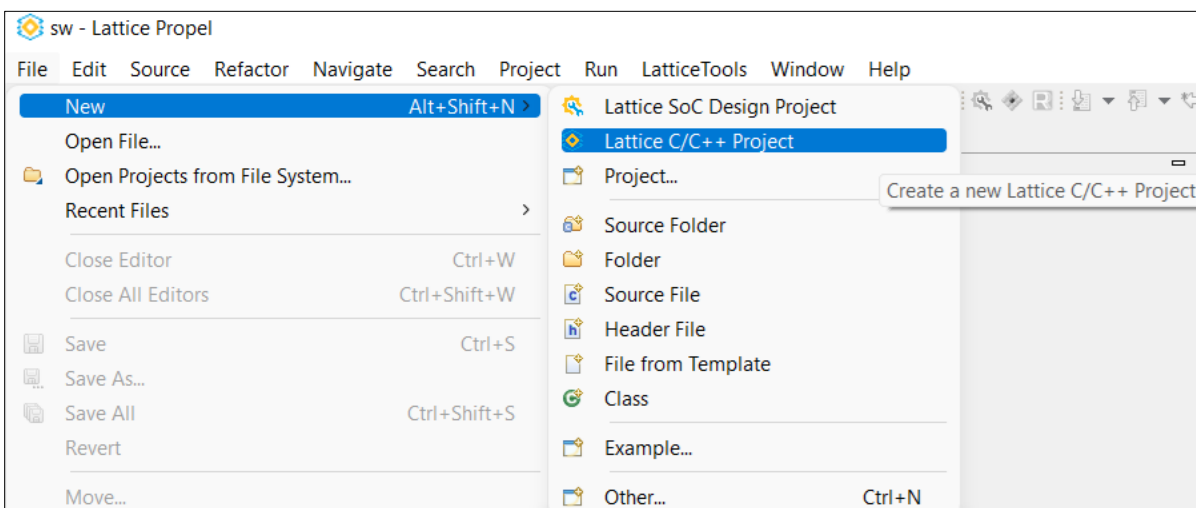


Figure 4.8. Lattice Propel SDK – Create New C/C++ Project

8. Load System and BSP Configuration.

- a. Browse to the sys_env.xml file generated by Lattice Propel Builder, as shown in Figure 4.9.
- b. Select **cpu0_inst** as the target processor.
- c. Enter a name for your project.
- d. Click **Next**. Then, click **Finish** to complete the setup.

Load System and BSP

Load lattice system environment file and BSP package

Select system environment file and BSP package

System env: C:\workspace\multi_cpu\sge\sys_env.xml Browse...

Select processor core to create C/C++ Project

Core selected: cpu0_inst

Project type: C

System information

Device Family	CPU Name	Instance Name
LAV-AT	riscv_mc	cpu0_inst
LAV-AT	riscv_mc	cpu1_inst
LAV-AT	riscv_mc	cpu2_inst
LAV-AT	riscv_mc	cpu3_inst
LAV-AT	riscv_mc	cpu4_inst
LAV-AT	riscv_mc	cpu5_inst

Project name: multi_cpu0

☒ Use default location

Location: C:\workspace\multi_cpu\sw\multi_cpu0 Browse...

Choose file system: default

☐ Build the project

☒ Create a debug launch configuration for OpenOCD

< Back Next > Finish Cancel

Figure 4.9. Load System and BSP – Select cpu0_inst

9. Repeat steps 7 and 8 for Remaining CPUs.

Repeat **Step 7** and **Step 8** for each of the remaining CPUs, from CPU1 to CPU5. Select the corresponding processor instance, such as cpu1_inst, cpu2_inst, and so on during each setup, as shown in [Table 4.2](#).

Table 4.2. Load System and BSP – Select Remaining CPUs

Core Selected	Project Name
cpu0_inst	multi_cpu0
cpu1_inst	multi_cpu1
cpu2_inst	multi_cpu2
cpu3_inst	multi_cpu3
cpu4_inst	multi_cpu4
cpu5_inst	multi_cpu5

10. Build all software projects.

After all CPU software projects, from CPU0 to CPU5, have been created:

Navigate to **Project > Build All** to compile all projects simultaneously (Figure 4.10).

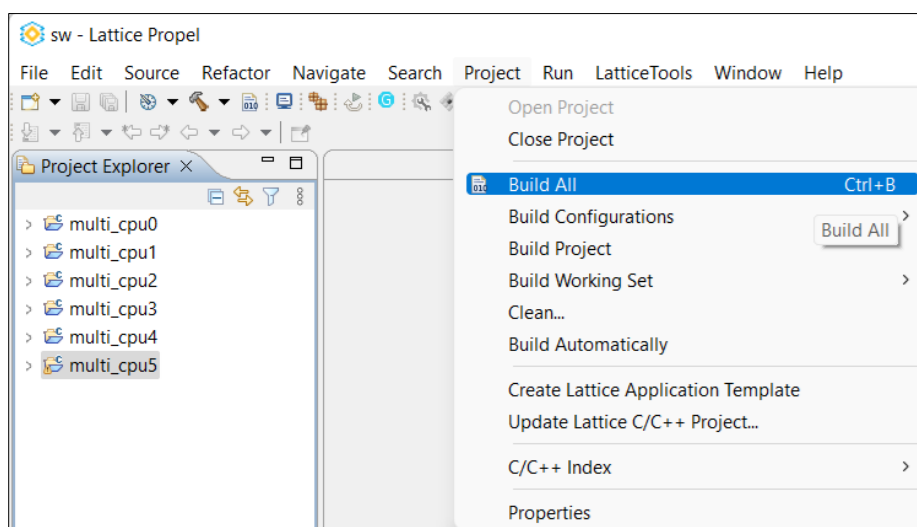


Figure 4.10. Lattice Propel SDK – Build All Projects

11. Verify the .mem file generation.

Ensure that a .mem file is generated for each project (Figure 4.11). These files should be located in the following directory: C:\workspace\multi_cpu\sw\multi_cpu*\Debug

Note: * represents CPU indices 0 through 5.

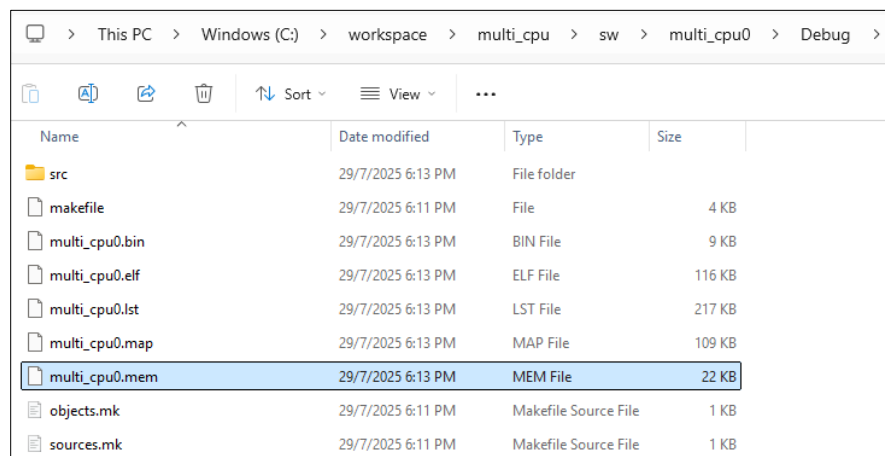


Figure 4.11. File Explorer – .mem Files for Each CPU Project

12. Initialize System Memory in Lattice Propel Builder (Figure 4.12).

In Lattice Propel Builder, initialize each system memory block (sysmem)* with its corresponding .mem file, as shown in Table 4.3.

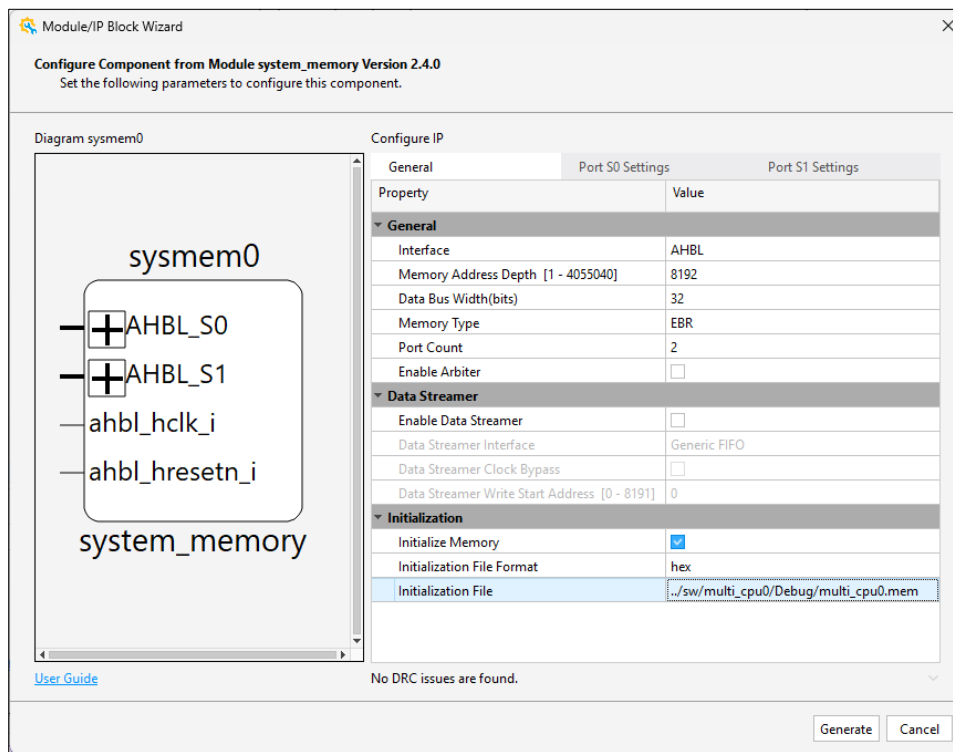


Figure 4.12. Lattice Propel Builder – Initialize System Memory with .mem Files

Table 4.3. Initialize System Memory with .mem Files

System Memory Instance Name	.mem File Name
sysmem0_inst	multi_cpu0.mem
sysmem1_inst	multi_cpu1.mem
sysmem2_inst	multi_cpu2.mem
sysmem3_inst	multi_cpu3.mem
sysmem4_inst	multi_cpu4.mem
sysmem5_inst	multi_cpu5.mem

13. Regenerate the SoC design.

After the memory initialization:

Click the **Generate** button again in Lattice Propel Builder to update the design with the new memory contents (Figure 4.13).

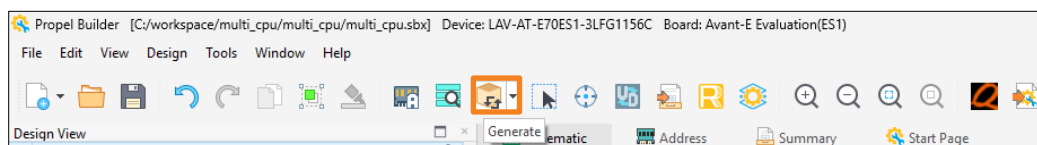


Figure 4.13. Lattice Propel Builder – Regenerate Design After Memory Initialization

14. Launch the Lattice Radiant software and compile the design.

Within Lattice Propel Builder:

- a. Launch the Lattice Radiant software.
- b. Compile the design to complete the hardware build process ([Figure 4.14](#)).

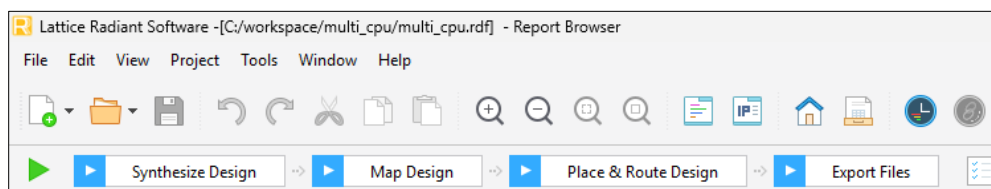


Figure 4.14. Launch Lattice Radiant – Compile Design

4.4. Running the Example Design

To view the output, set up a UART terminal as described in the instructions below. The bitstream file generated in the [Generating the Example Design](#) section is downloaded to the Avant-E Evaluation Board using the Lattice Radiant Programmer.

4.4.1. Setting Up the UART Terminal

The software code in this example design displays messages on a terminal through the UART interface. To view the output, follow steps below to set up a UART terminal:

1. Launch Tera Term, or use Serial Terminal in SDK, or PuTTY.
2. Go to **File > New Connection**.
3. Select **Serial**.
4. Choose the correct COM port, which is usually the higher number.
5. Click **OK**.
6. Go to **Setup > Serial Port** and configure:
 - Baud Rate: 115200
 - Data: 8 bits
 - Parity: None
 - Stop Bits: 1
 - Flow Control: None
 - Transmit Delay: 0 msec/char; 0 msec/line
7. Click **New Setting** to complete the setup process.

4.4.2. Programming the FPGA Bitstream

To program the bitstream into the device, follow these steps:

1. Launch Radiant Programmer.
Open the Lattice Radiant software, then go to **Tools > Programmer** ([Figure 4.15](#)).

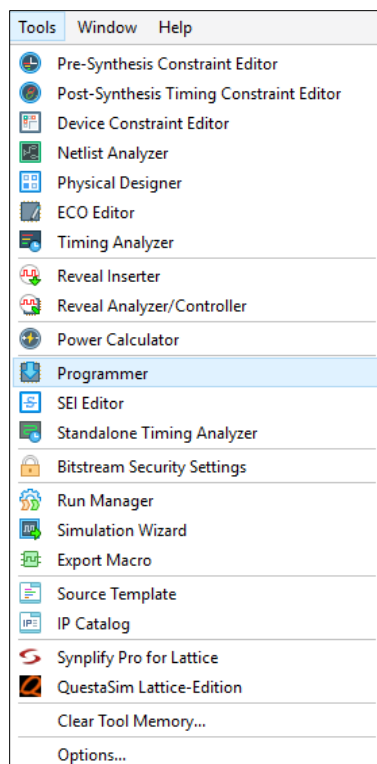


Figure 4.15. Launching Radiant Programmer

2. Configure the programmer settings.

In the Lattice Radiant Programmer window:

- Ensure the **Device Family**, **Device**, and **Operation** match the configuration shown in Figure 4.16.
- Select the .bit file generated from previous steps using the Lattice Radiant software.

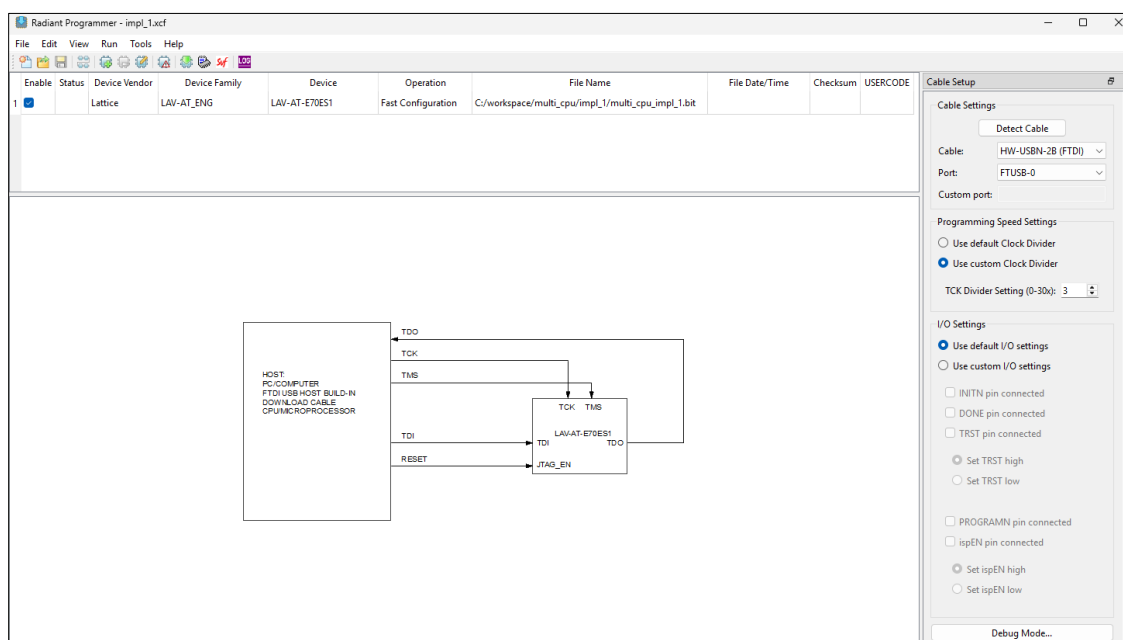


Figure 4.16. Lattice Radiant Programmer – Device and Bitstream Configuration

3. Detect the programming cable.
In the **Cable Setup** window, click **Detect Cable** (Figure 4.17).

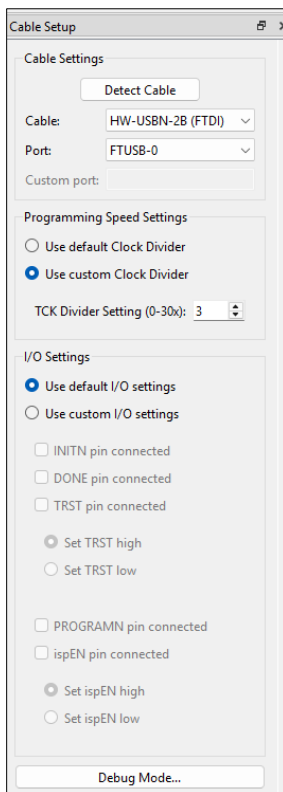


Figure 4.17. Cable Setup – Detecting Cable

4. Select the programming cable.
In the **Select Cable** window:
 - a. Choose **FTUSB-0** from the list.
 - b. Click **OK** to confirm (Figure 4.18).

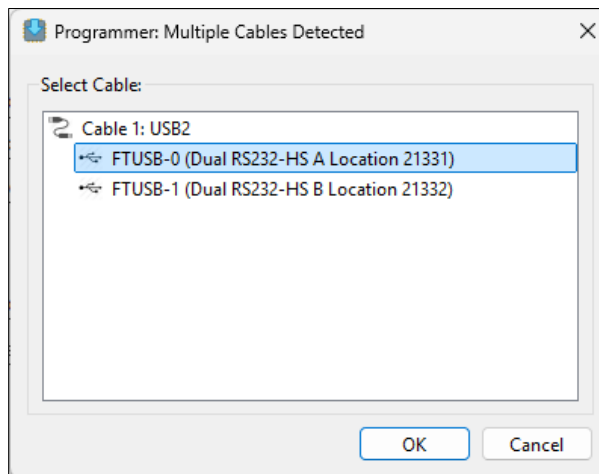


Figure 4.18. Select Cable – FTUSB-0 Option

5. Program the device.

In the Lattice Radiant Programmer main interface:

- a. Go to **Run > Program Device**.
- b. The **Output Console** displays a message indicating that the operation is successful (Figure 4.19).

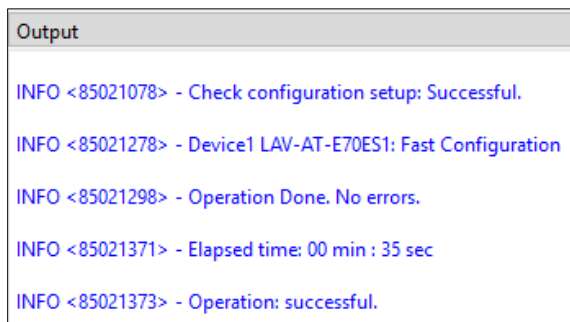


Figure 4.19. Output Console – Programming the FPGA Bitstream

4.4.3. Verifying Results

Verify the UART terminal output.

After programming the FPGA and running the design, check the UART terminal, such as Tera Term, to ensure the printed messages match the expected output, as shown in Figure 4.20.

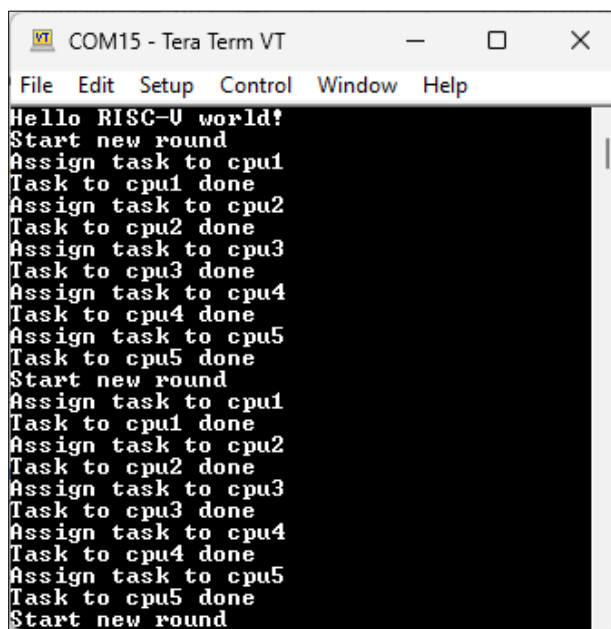


Figure 4.20. UART Terminal – Expected Output Messages

Appendix A. Resource Utilization

Table A.1. Resource Utilization in CertusPro-NX-100 Device

Configuration	LUTs	Registers
Mutex Number = 2, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	67	34
Mutex Number = 2, Port Number = 2, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	71	36
Mutex Number = 16, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	329	104
Mutex Number = 16, Port Number = 2, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	402	120
Mutex Number = 16, Port Number = 8, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	1805	238
Mutex Number = 16, Port Number = 8, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	1966	313
Mutex Number = 2, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AXI-L	85	60
Mutex Number = 16, Port Number = 8, Port Protection = 0, CPU ID Width = 4, Interface Type = AXI-L	1805	327
Mutex Number = 16, Port Number = 8, Port Protection = 1, CPU ID Width = 4, Interface Type = AXI-L	2308	404

Table A.2. Resource Utilization in LAV-AT-E70 Device

Configuration	LUTs	Registers
Mutex Number = 2, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	56	34
Mutex Number = 2, Port Number = 2, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	60	36
Mutex Number = 16, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	330	118
Mutex Number = 16, Port Number = 2, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	398	140
Mutex Number = 16, Port Number = 8, Port Protection = 0, CPU ID Width = 4, Interface Type = AHB-L	1694	240
Mutex Number = 16, Port Number = 8, Port Protection = 1, CPU ID Width = 4, Interface Type = AHB-L	1948	289
Mutex Number = 2, Port Number = 2, Port Protection = 0, CPU ID Width = 4, Interface Type = AXI-L	77	60
Mutex Number = 16, Port Number = 8, Port Protection = 0, CPU ID Width = 4, Interface Type = AXI-L	1623	339
Mutex Number = 16, Port Number = 8, Port Protection = 1, CPU ID Width = 4, Interface Type = AXI-L	2107	407

References

- [Lattice Propel Builder 2025.2 User Guide \(FPGA-UG-02243\)](#)
- [Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide \(FPGA-UG-02039\)](#)
- [AMBA AXI and ACE Protocol Specification](#)

For more information, refer to:

- [Lattice Propel](#) web page
- [Lattice 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
- [Lattice Insights for 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 v1.0.0, December 2025

Section	Change Summary
All	Production release.



www.latticesemi.com