I/O Aggregation Over USB with CrossLinkU-NX Reference Design User Guide

Reference Design

FPGA-RD-02288-2.0

December 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
AHB	Advanced High-Performance Bus
AHB-Lite	Advanced High-Performance Bus Lite version
APB	Advanced Peripheral Bus
AXI	Advanced Extensible Interface
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
I2C	Inter-Integrated Circuit; A synchronous, multi-controller, multi-target, packet switched, single-ended, serial bus
I/O	Input/Output
I/O-DB	Input/Output Daughter Board
IP	Intellectual Property
LED	Light Emitting Diode
LMMI	Lattice Memory Mapped Interface
PC	Personal Computer
PLL	Phase-Locked Loop
RISC-V	Reduced Instruction Set Computer Five
RTL	Register Transfer Level
SDK	Software Development Kit
SPI	Serial Peripheral Interface
TRB	Transfer Request Block
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

1. Introduction

The Lattice Semiconductor I/O Aggregation Over USB2 reference design provides developers a template to bridge USB to several interfaces defined below from a PC (Windows or Linux).

1.1. Quick Facts

Download the reference design files from the Lattice USB to I/O Aggregation and Bridging Reference Design web page.

Table 1.1. Summary of the Reference Design

General	Target Device	LIFCL-33U
	Source Code Format	C, RTL
Simulation/Validation	Functional Simulation	Not supported
	Timing Simulation	Not supported
	Hardware Validation	Fully validated
Software Requirements	Software Tool and Version	Lattice Propel SDK 2024.2 Lattice Propel Builder 2024.2 Lattice Radiant Software Version 2024.2 Lattice Radiant Programmer Version 2024.2
	IP Version	RISC-V MC Version 2.7.0 System Memory Version 2.3.0 AHB Lite Interconnect Version 1.3.2 AHB Lite to APB Bridge Version 1.1.2 AHB-Lite Feedthrough Version 1.0.0 APB Interconnect Version 1.2.1 GPIO Version 1.6.2 I2C Controller Version 2.0.1 SPI Controller Version 2.1.0 UART Version 1.3.0
Hardware Requirements	Board	LIFCL-33U-Evaluation Board REV-B
	Cable	USB C to USB C or USB C to USB A (9 pins) Cable
	EEPROM Memory Module	AT24C256 256k Bits EEPROM Memory Module with I2C Interface
	USB to UART Adapter	DSD TECH SH-U05A USB to UART Adapter

1.2. Features

Key features of the I/O Aggregation Over USB reference design include:

- Hardwire USB supports I/O aggregation over USB.
- Wrapper RTL includes RISC-V, System Memory, and AHB bridge for USB enumeration.
- Maximum 8 endpoints can be flexibly configured as user selected peripheral:
  - GPIO Input
  - GPIO Output
  - I2C Controller
  - SPI Controller
- Windows drivers and Python script to communicate to the peripherals.

1.3. Naming 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

2. Directory Structure and Files

Figure 2.1 shows the directory structure.

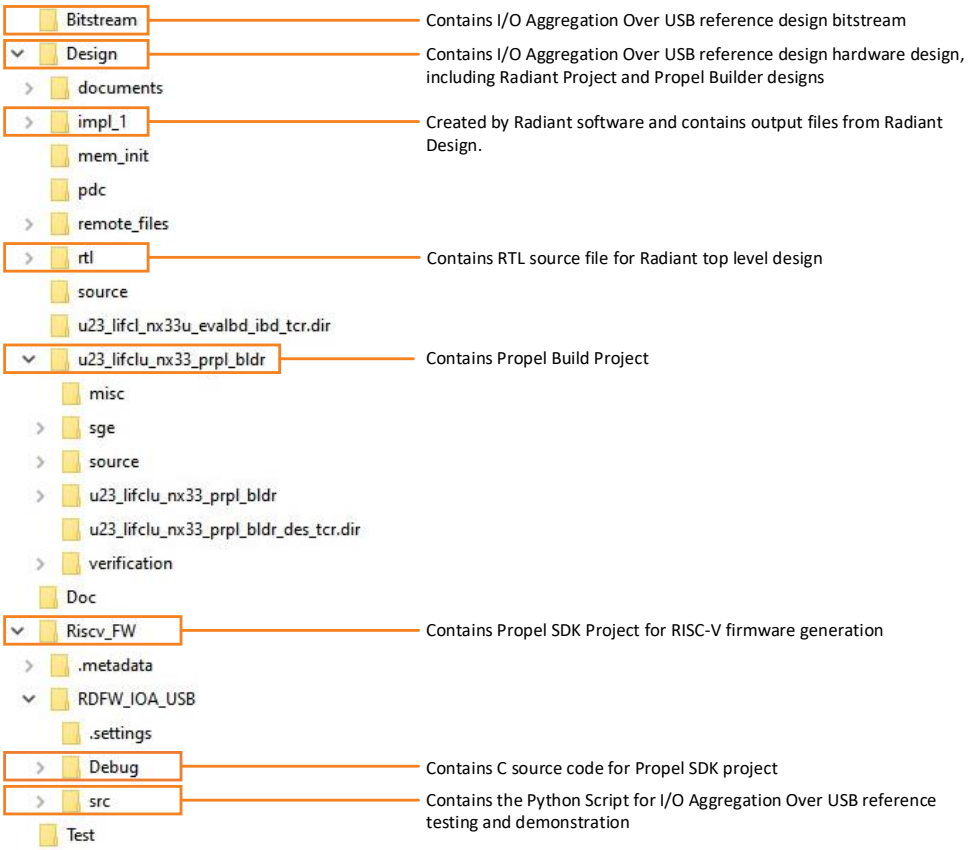

Figure 2.1. Directory Structure

The RD_IOA_USB2.zip package includes:

- Propel SDK project, which handles the C code project for RISC-V firmware generation and compiling.
- Propel Build project, which contains the RISC-V microcontroller project, including peripheral soft IPs such as GPIO, I2C Controller, and SPI Controller.
- Radiant project for the top level design and bitstream generation, which contains the design block exported from the Propel Builder Project, USB23 Hard IP, PLL for clock handling, and others.

Table 2.1 shows the list of files included in the reference design package.

Table 2.1. File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated to the generated IP.
<Component name>.cfg	This file contains the parameter values used in IP configuration.
component.xml	Contains the ipxact:component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	This file provides an example RTL top file that instantiates the module.
rtl/<Component name>_bb.v	This file provides the synthesis closed box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the module.

3. Functional Description

The top level block diagram of the I/O Aggregation Over USB2 reference design is shown in Figure 3.1 below.

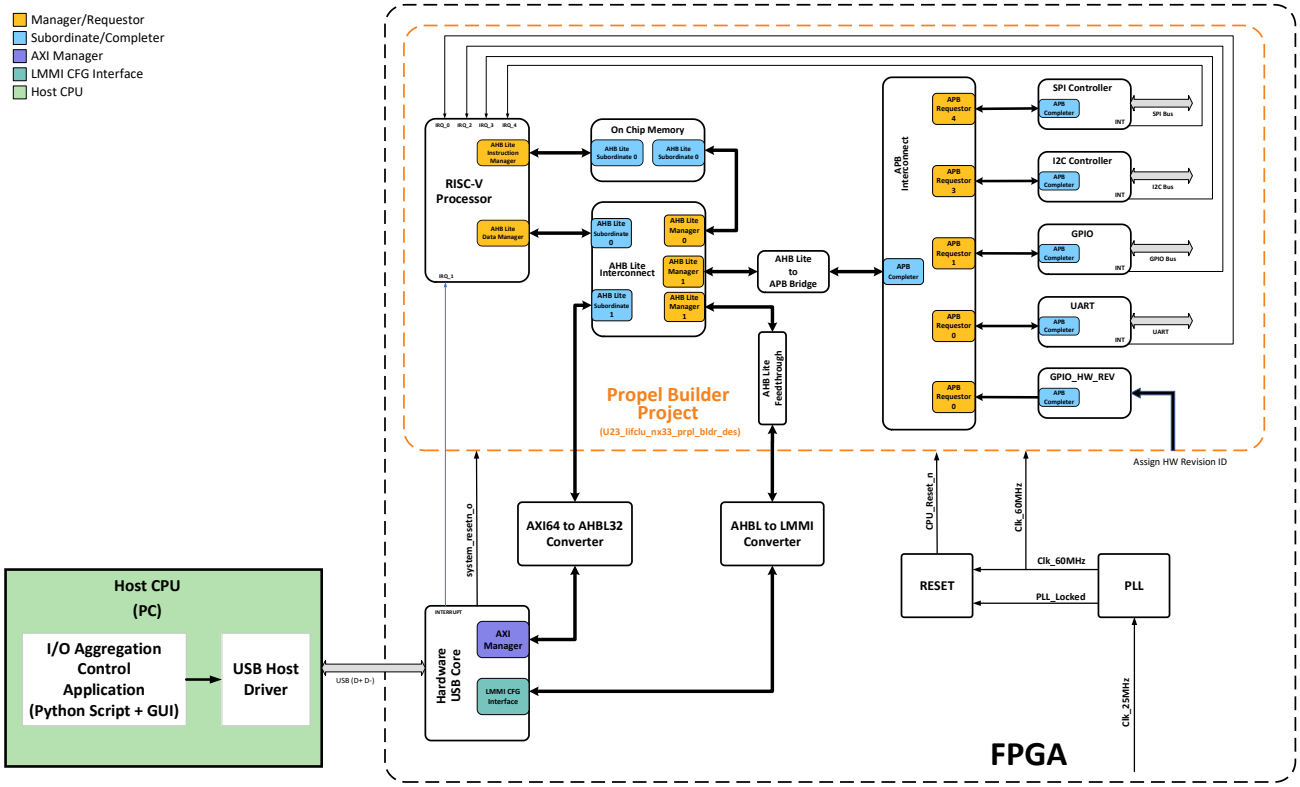

Figure 3.1. Reference Design Block Diagram

3.1. Design Components

The I/O Aggregation Over USB2 reference design includes the following blocks:

- u23_lifclu_nx33_prpl_bldr_des which includes:
  - RISC-V microcontroller for USB enumeration and peripheral control
  - Peripheral soft IP such as GPIO, I2C Controller, SPI Controller, and UART
- USB23 Hardware Primitive

3.2. Clocking Scheme

The I/O Aggregation Over USB2 reference design uses a single clock at 60 MHz for all sub-blocks, including the RISC-V processor. A PLL is used to generate the 60 MHz clock from a 25 MHz clock on the LIFCL-33U Evaluation Board.

3.2.1. Clocking Overview

The clock domain block diagram is shown in Figure 3.2.

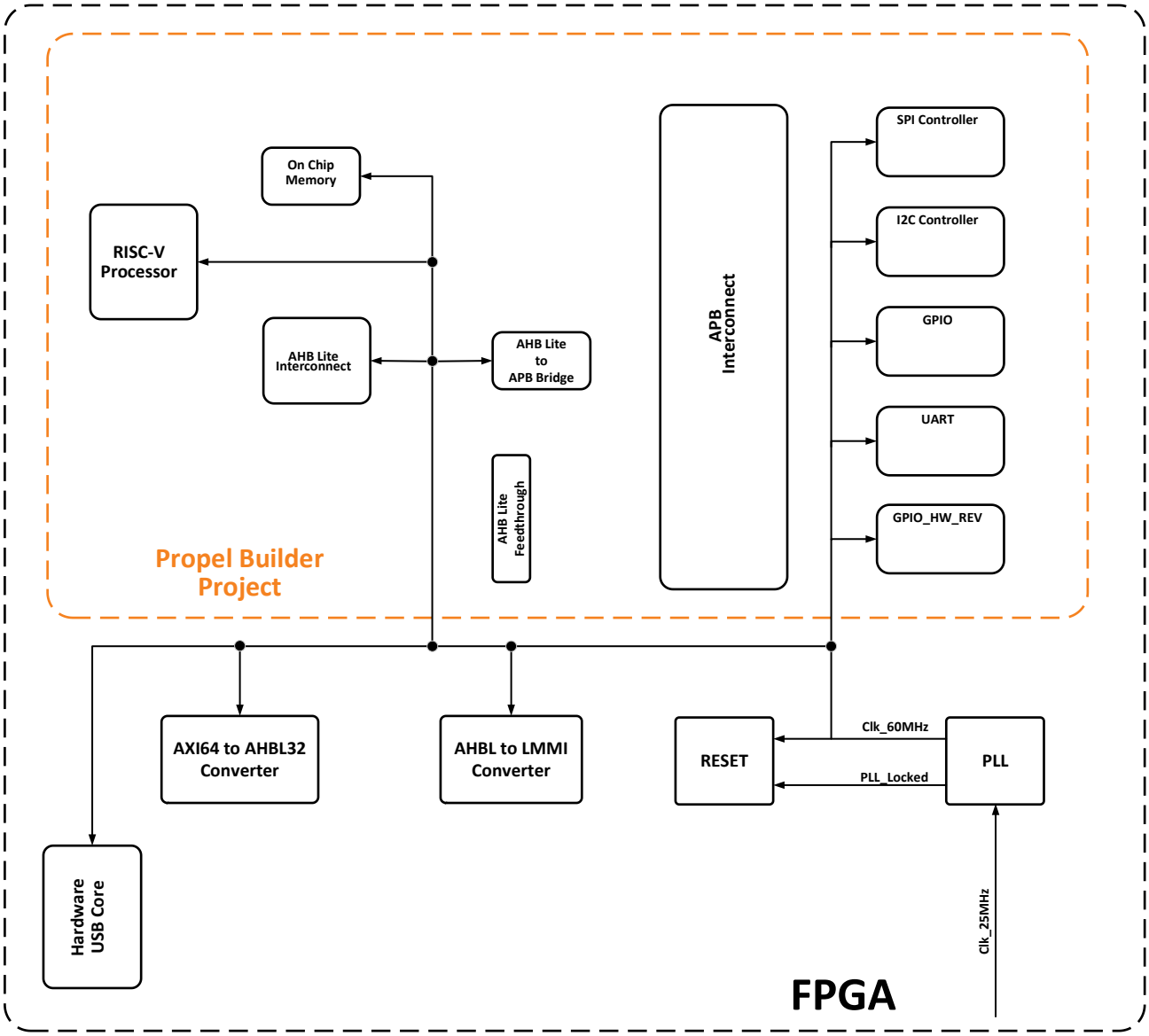

Figure 3.2. Reference Design Clock Domain Block Diagram

3.3. Reset Scheme

An active low synchronous reset is generated from the PLL clock and PLL lock output. This reset is sent to the RISC-V microcontroller. Then, the system_resetn_o from the RISC-V microcontroller is distributed over the whole reference design.

3.3.1. Reset Overview

The block diagram of reset scheme for this reference design is shown in Figure 3.3.

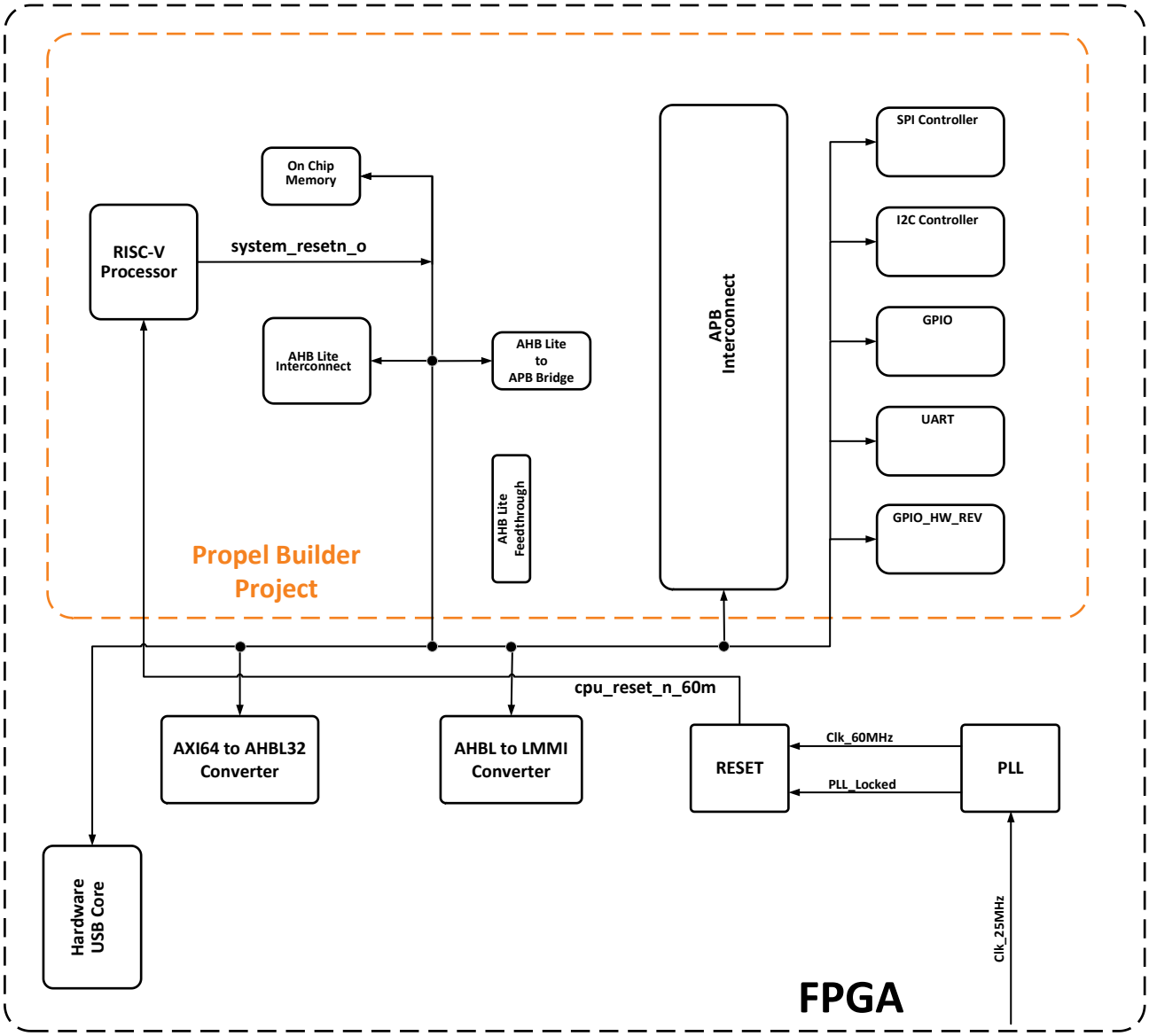

Figure 3.3. Reference Design Reset Scheme Block Diagram

4. Signal Description

The input/output interface signals for the I/O Aggregation Over USB reference design are shown in Table 4.1.

Table 4.1. Primary I/O

Parameter	Direction	LIFCL-33U Ball	Eval. Board Access	I/O-DB Access	Description
clk_25m_i	Input	G7	—	—	Reference clock input for the PLL to generate the 60 MHz system clock. The clock frequency is 25 MHz from the Crystal oscillator on the LIFCL-33U Evaluation Board.
REFINCLKEXTP_i	Input	F8	—	—	USB3.0 PHY external positive differential clock
REFINCLKEXTM_i	Input	E8	—	—	USB3.0 PHY external negative differential clock
dp_z	Input/Output	D7	—	—	USB2.0 positive differential data line
dm_z	Input/Output	E7	—	—	USB2.0 negative differential data line
u3_rxp_i	Input	A8	—	—	USB3.0 positive differential data line for receiver
u3_rxm_i	Input	B8	—	—	USB3.0 negative differential data line for receiver
u3_txp_o	Output	A7	—	—	USB3.0 positive differential data line for transmitter
u3_txm_o	Output	A6	—	—	USB3.0 negative differential data line for transmitter
vbus_z	Input/Output	E5	—	—	Power signal
gpio_0_z[0]	Output	G6	LED D5	—	GPIO output
gpio_0_z[1]	Input	L4	CN3 Pin 9	PMOD J1 Pin 7	GPIO output
i2cm_scl	Input/Output	M5	CN3 Pin 7	PMOD J1 Pin 3	Peripheral I2C Controller SCL line
i2cm_sda	Input/Output	M4	CN3 Pin 8	PMOD J1 Pin 4	Peripheral I2C Controller SDA line
spim_sclk_o	Output	B4	J12 Pin 8	—	Peripheral SPI Controller clock output
spim_ssn_o	Output	B3	J12 Pin 1	—	Peripheral SPI Controller Chip Select output
spim_mosi_o	Output	D4	J12 Pin 3	—	Peripheral SPI Controller MOSI output
spim_miso_i	Input	D3	J12 Pin 5	—	Peripheral SPI Controller MISO input
uart_txd_o	Output	J2	CN3 Pin 18	PMOD J2 Pin 7	UART output for RISC-V microcontroller debugging
uart_rxd_i	Input	H2	CN3 Pin 19	PMOD J2 Pin 8	UART input for RISC-V microcontroller debugging

5. Building the Reference Design

This section describes how to run the I/O Aggregation Over USB2 reference design using the Lattice Radiant software.

For more details on the Lattice Propel Software and Lattice Radiant software, refer to the Lattice Propel Software User Guide and Lattice Radiant Software User Guide.

5.1. Running Propel SDK Project

The I/O Aggregation Over USB reference design utilizes the hardened USB block inside the LIFCL-33U which needs to be configured via RISC-V firmware. This section describes how to build and run a C project in the Lattice Propel software tool.

5.1.1. Opening Propel SDK Project

1. To open the Propel SDK project in the Lattice Propel software tool, launch the Lattice Propel software tool.
2. Click the **Browse** button to set workspace folder to *.\Riscv_FW*, then click the **Launch** button, as shown in Figure 5.1.

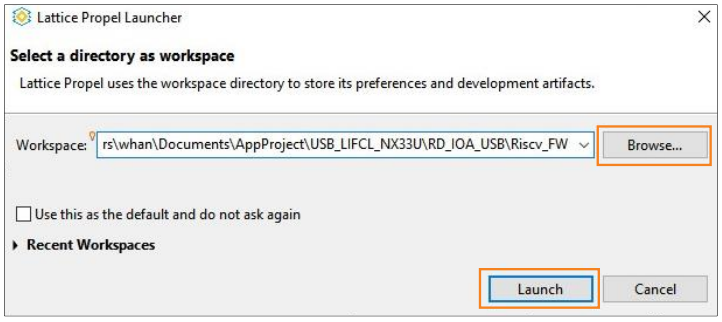

Figure 5.1. Launch Lattice Propel Software Tool

3. The Propel SDK project is opened with **RDFW_IOA_USB** project as shown in Figure 5.2.

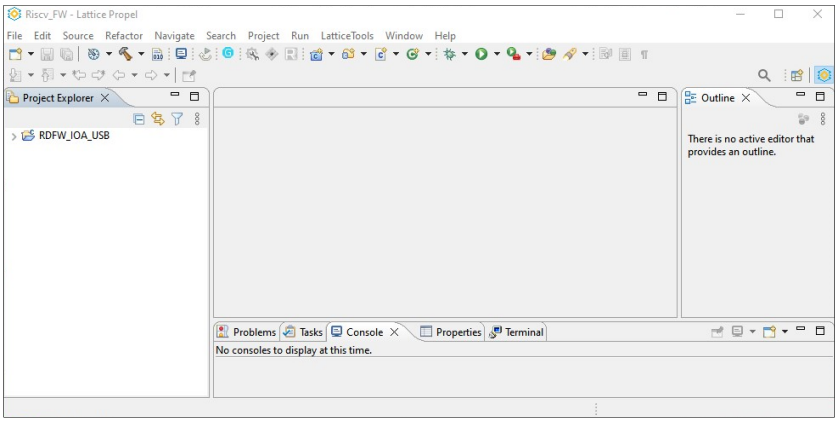

Figure 5.2. Propel SDK Project in Lattice Propel

5.1.2. Navigating Propel SDK Project

In Propel Project Explorer, navigate to **RDFW_IOA_USB** > **src** to expand the project file list for reviewing purposes as shown in Figure 5.3.

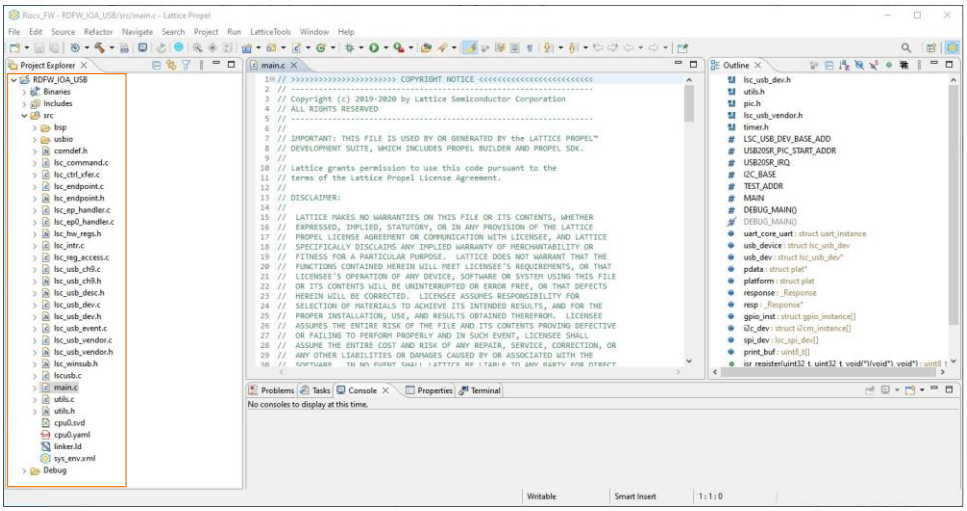

Figure 5.3. Navigating the Propel SDK Project

The I/O aggregation requests are processed in the **lsc_usb_vendor.c**. The request process flow is shown in Figure 5.4.

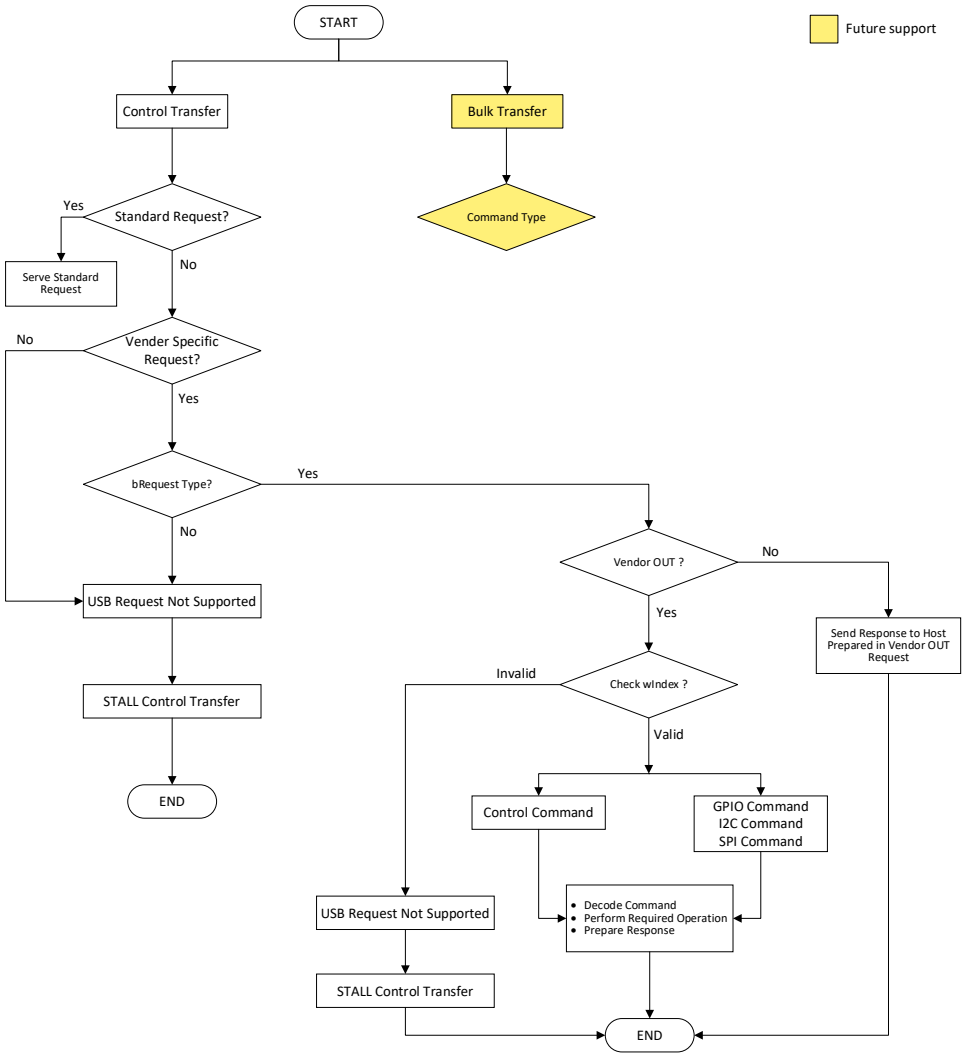

Figure 5.4. I/O Aggregation Request Process Flow

5.1.3. Generating Output MEM file

1. To compile the Propel SDK project to generate the memory image for the RISC-V firmware, select the Propel Build Configuration Active to **Debug** mode, as shown in Figure 5.5.

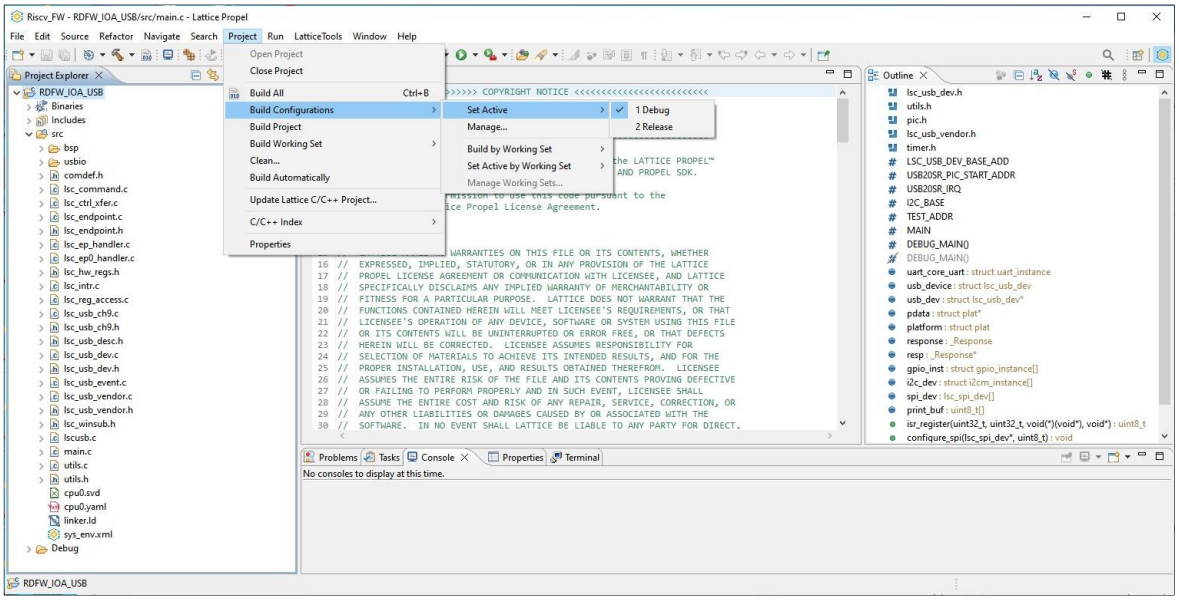

Figure 5.5. Set Propel Build Configuration Mode

2. To launch the project building process, right-click on the process name **RDFW_IOA_USB**, select **Clean Project** from the pull down menu to clean up the output folders. Then select the **Build Project** from the pull down menu, as shown in Figure 5.6.

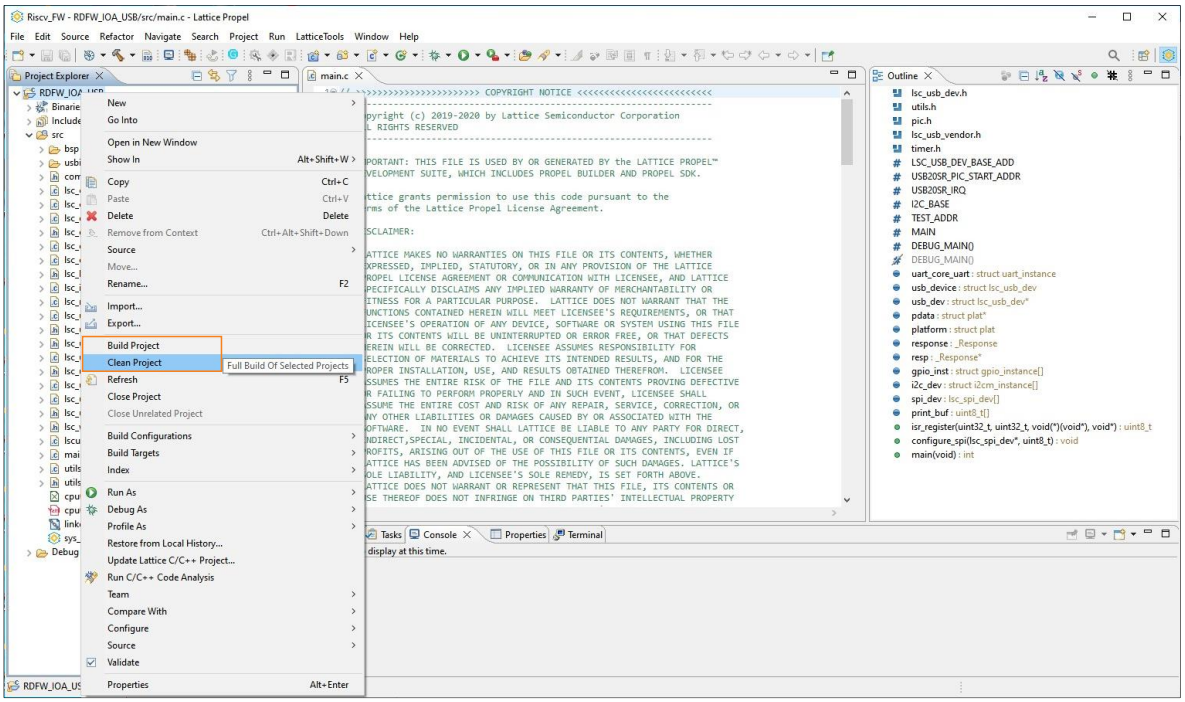

Figure 5.6. Launch the Project Building Process

3. The output **RDFW_IOA_USB.mem** file can be located at: *\Riscv_FW RDFW_IOA_USB\Debug\RDFW_IOA_USB.mem*.

5.2. Running Propel Builder Design

The I/O Aggregation Over USB reference design utilizes the hardened USB block inside the LIFCL-33U which requires RISC-V microcontroller to handle the USB hard IP enumeration and USB traffic control. This section describes how to evaluate the RISC-V design, including the peripheral soft IPs, in the Lattice Propel Builder software tool.

5.2.1. Opening Propel Builder Design

To open the Propel Builder project in Lattice Propel Builder software tool, launch the Lattice Propel software tool. Click on **File** > **Open Design** and navigate through the **Open sbx** window to open the *.\Design\ u23_lifclu_nx33_prpl_bldr\ u23_lifclu_nx33_prpl_bldr\ u23_lifclu_nx33_prpl_bldr_des.sbx* as shown in Figure 5.7.

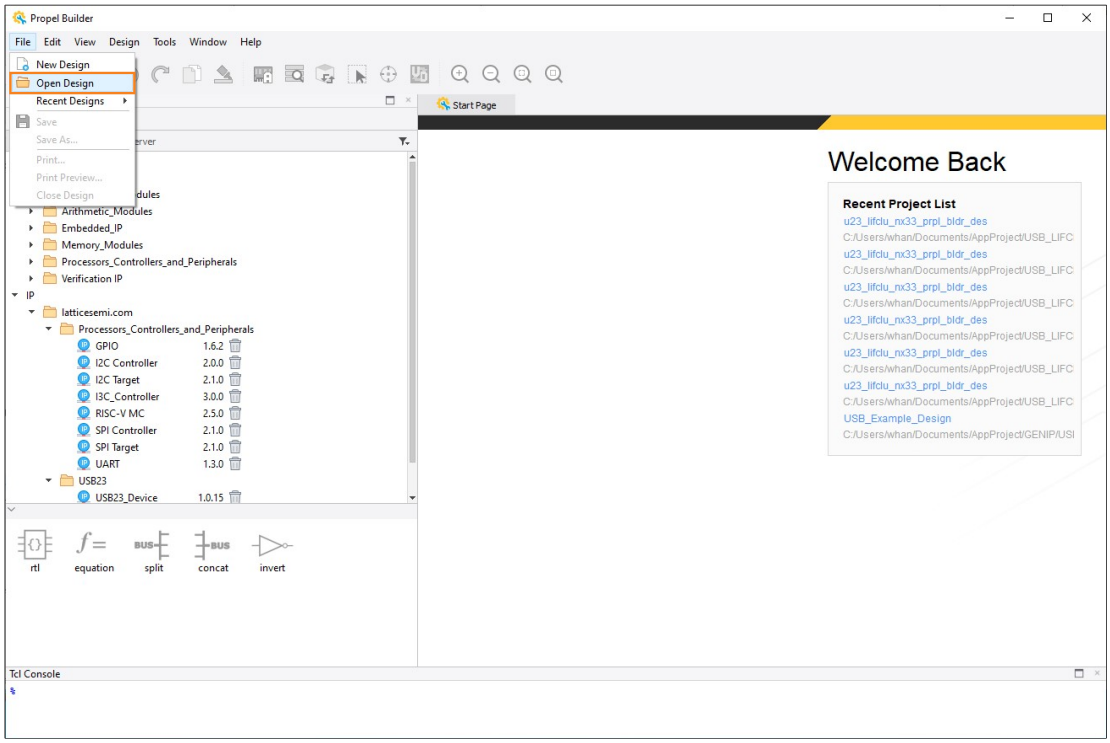

Figure 5.7. Open the Propel Build Design

5.2.2. Mapping Design with Firmware

1. In the Propel Build design schematic, double-click on the **sysmem0_inst** to open the system_memory Module Block Wizard, as shown in Figure 5.8.

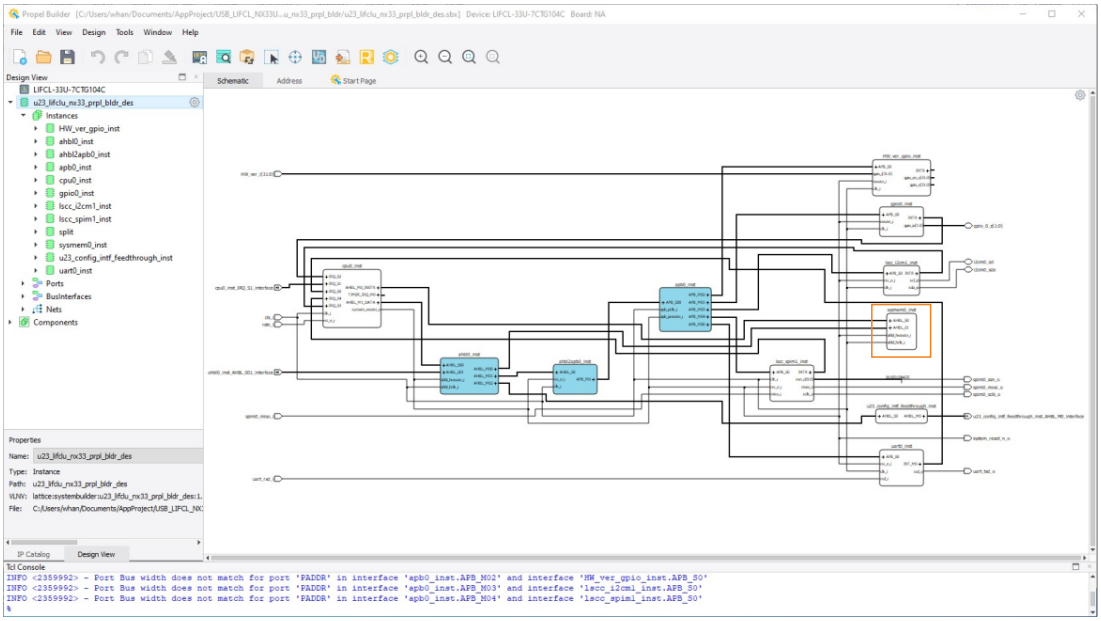

Figure 5.8. Opening System Memory Module Block Wizard

2. In the system memory Module Block Wizard, make sure the Initialization File is mapped to *../../../Riscv_FW/USB23_Broad_Market_1.7/Debug/USB23_Broad_Market.mem* or the latest revision of the mem file, then click on the **Generate** button, as shown in Figure 5.9.

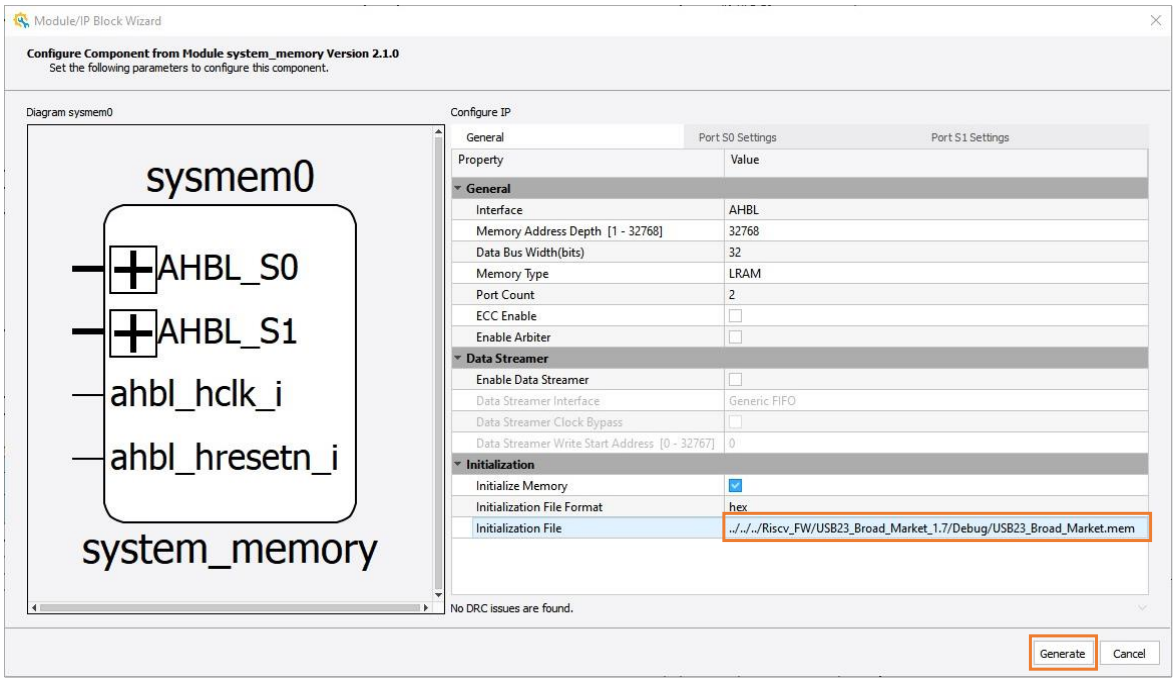

Figure 5.9. Mapping the System Memory Initialization File

3. Click the **Address** tab to review the base address assignment for the system and all peripherals within the Propel Builder Project, as shown in Figure 5.10.

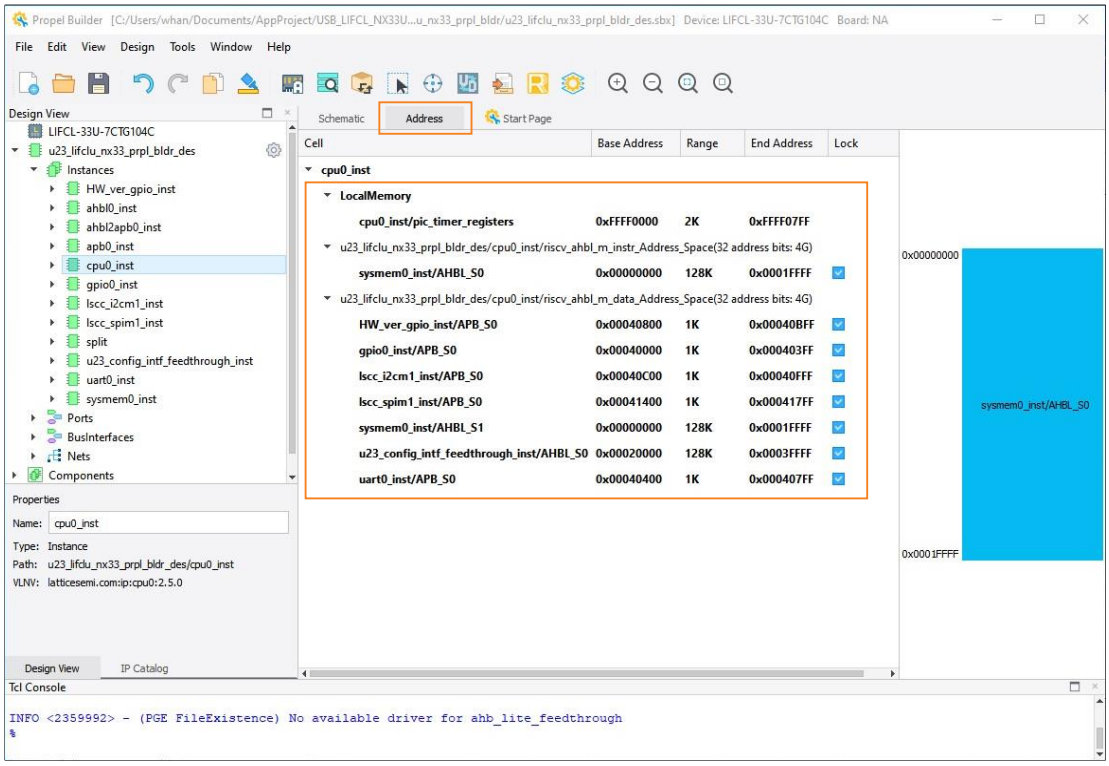

Figure 5.10. Base Address Assignment

5.2.3. Exporting Design to Radiant Software

To export the propel builder design to the Lattice Radiant software, click on the **Generate** icon as shown in Figure 5.11.

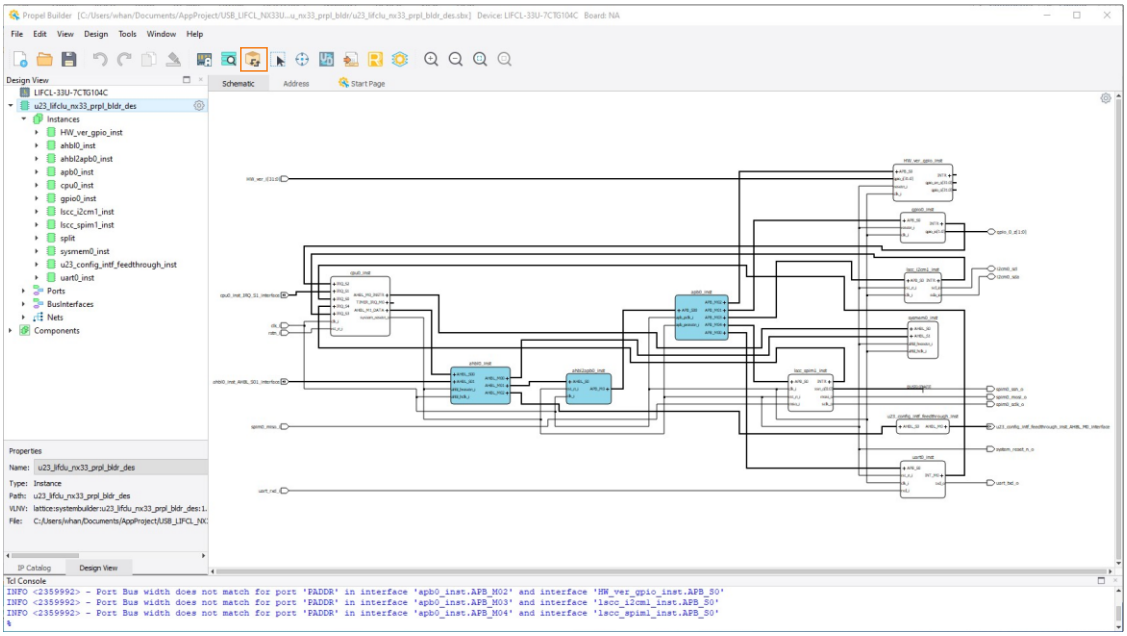

Figure 5.11. Export the Propel Builder Design to Radiant Software

The exported files are located in the *./Design/u23_lifclu_nx33_prpl_bldr/u23_lifclu_nx33_prpl_bldr* folder.

5.3. Running Radiant Project

5.3.1. Opening Radiant Project

This section provides the procedure of creating your FPGA bitstream file using the Lattice Radiant Software.

1. To create the FPGA bitstream file, open the Lattice Radiant software. Then, click on **Open Project** icon, as shown in Figure 5.12.

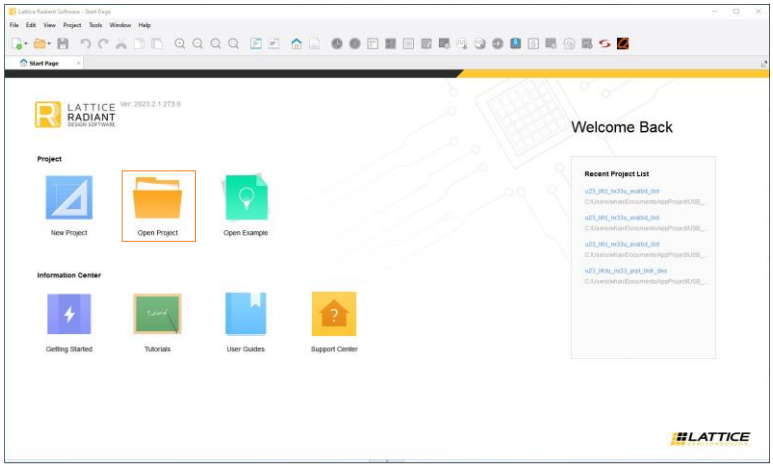

Figure 5.12. Lattice Radiant Software

2. Open the Radiant project file *u23_lifcl_nx33u_evalbd_ibd.rdf* from the *./Design/* folder, as shown in Figure 5.13. The design should have no errors but there are 16 warnings due to some unused signals from some instance. The warnings should not affect the design compilation and bitstream generation.

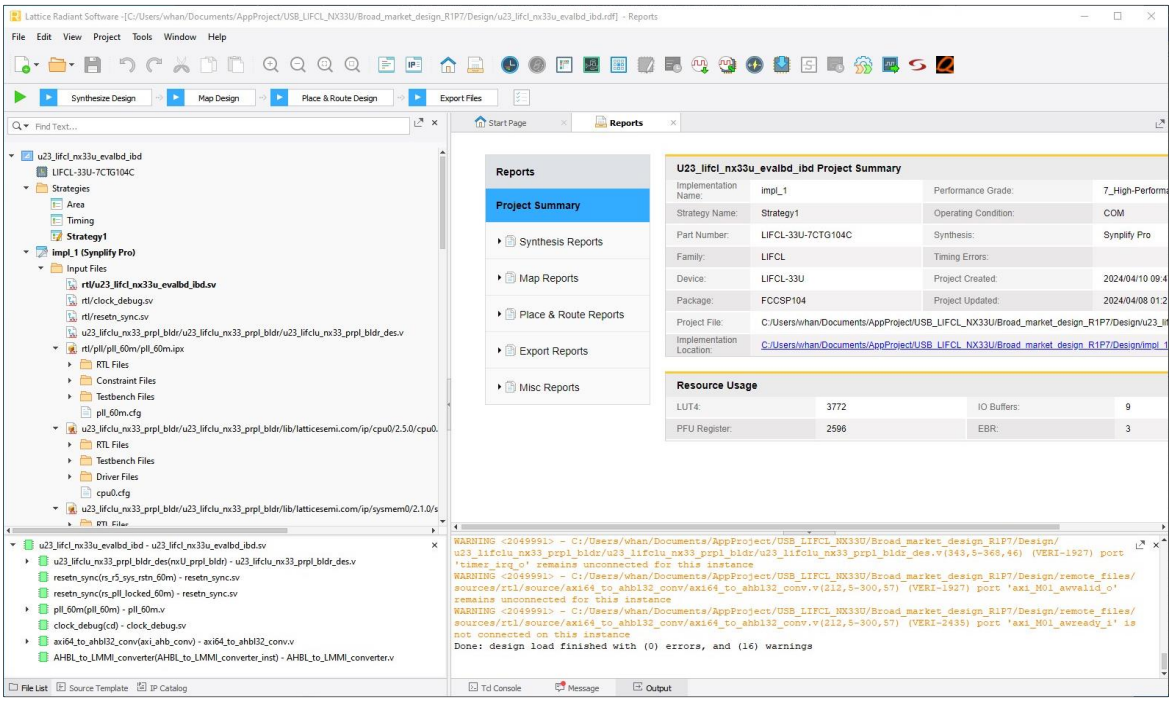

Figure 5.13. Open the Radiant Design

5.3.2. Generating Bitstream File

Click **Export Files** to generate the bitstream file. View the log message from the *./Design/impl_1* folder.

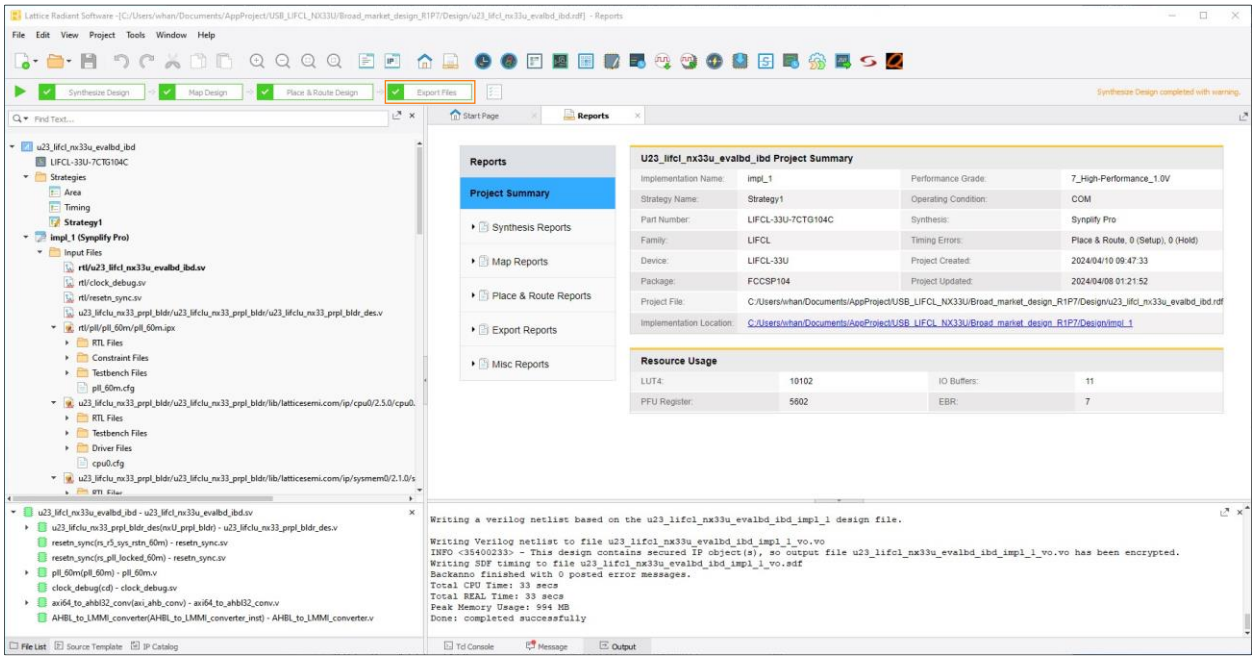

Figure 5.14. Generated Bitstream Log

The generated bitstream could be located as *./Design/impl_1/ u23_lifcl_nx33u_evalbd_ibd_impl_1.bit*.

6. LIFCL-33U Evaluation Board Programming

In case the LIFCL-33U Evaluation Board needs to be programmed or re-programmed, refer to the steps in the following subsections.

6.1. LIFCL-33U Evaluation Board Connection for Programming

Connect the Micro USB port (J2) of LIFCL-33U Evaluation Board to your PC by using a Micro USB cable.

6.2. Radiant Programmer GUI Setup

1. Launch the Radiant Programmer. Navigate to the **Edit** > **Device Properties…** menu item or click the **Operation** area to bring up the **Device Properties** window as shown in Figure 6.1.

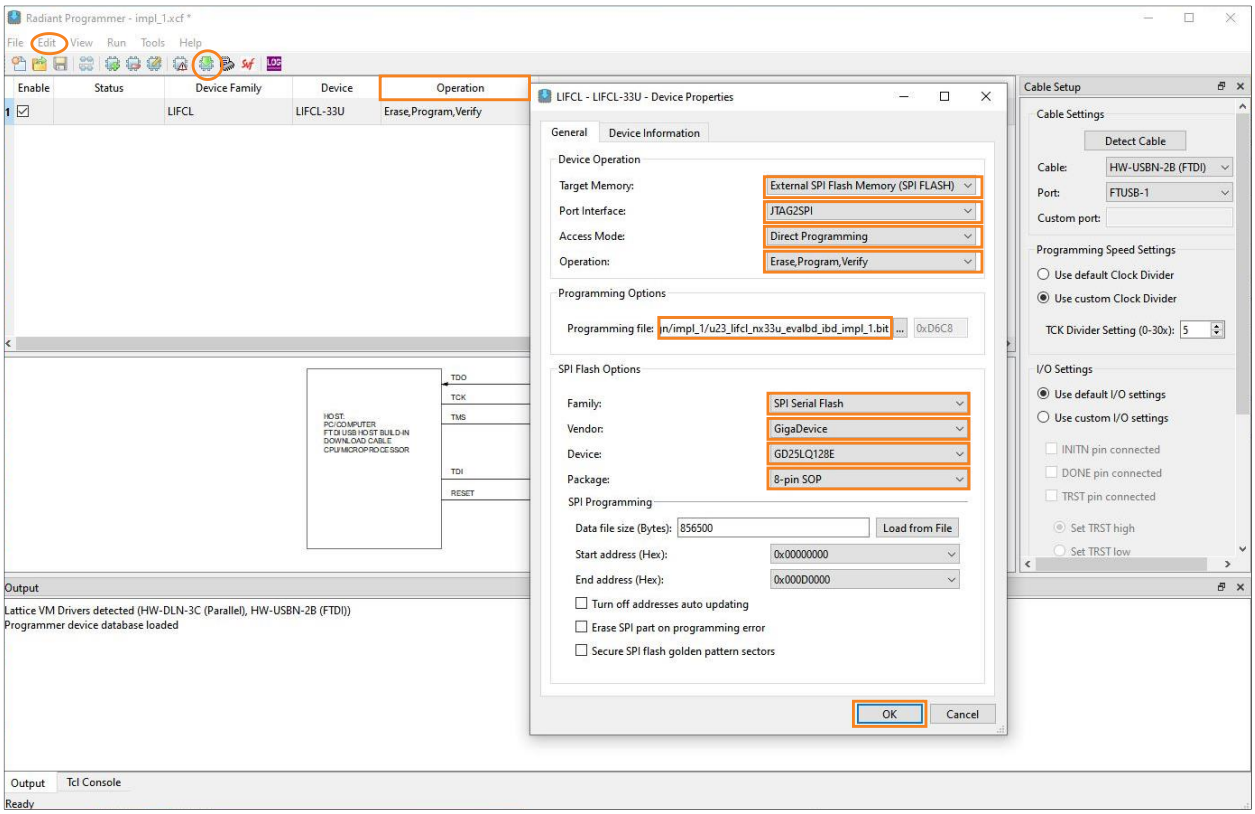

Figure 6.1. Radiant Programmer GUI

2. Select your desired **Target Memory**, **Port Interface**, **Access Mode,** and **Operation**.
3. Select the bitstream file from the **Bitfiles** folder and choose the **SPI Flash Options** as shown in Figure 6.1.
4. Click **OK** when finished.

6.3. Programming the Evaluation Board

Click the **Program Device** icon from the toolbar to perform the device programming as shown in Figure 6.2.

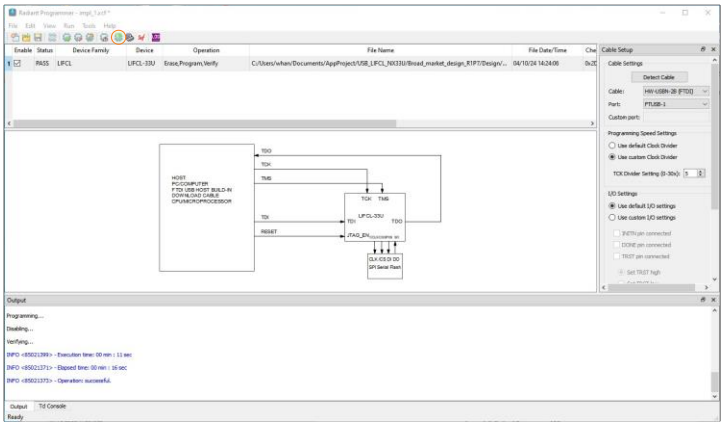

Figure 6.2. Program the Device

7. Running the Reference Design on Evaluation Board

This section describes how to run the I/O Aggregation Reference Design Demo on the LIFCL-33U Evaluation Board. For more details on the LIFCL-33U Evaluation Board, contact Lattice Sales to obtain the board document.

7.1. LIFCL-33U Evaluation Board Connection to PC

Connect the USB Type C connector (CN2) on the pre-programmed LIFCL-33U Evaluation Board to your PC using the USB type C to USB type C cable or the USB type C to USB type A (9 pins) cable.

7.2. Installing libusb-win32 Driver for USB23 on LIFCL-33U Evaluation Board

By default, the USB on LIFCL-33U is assigned with a WINUSB driver which is shown under standard Universal Serial Bus devices in Windows Device Manager, as shown in Figure 7.1.

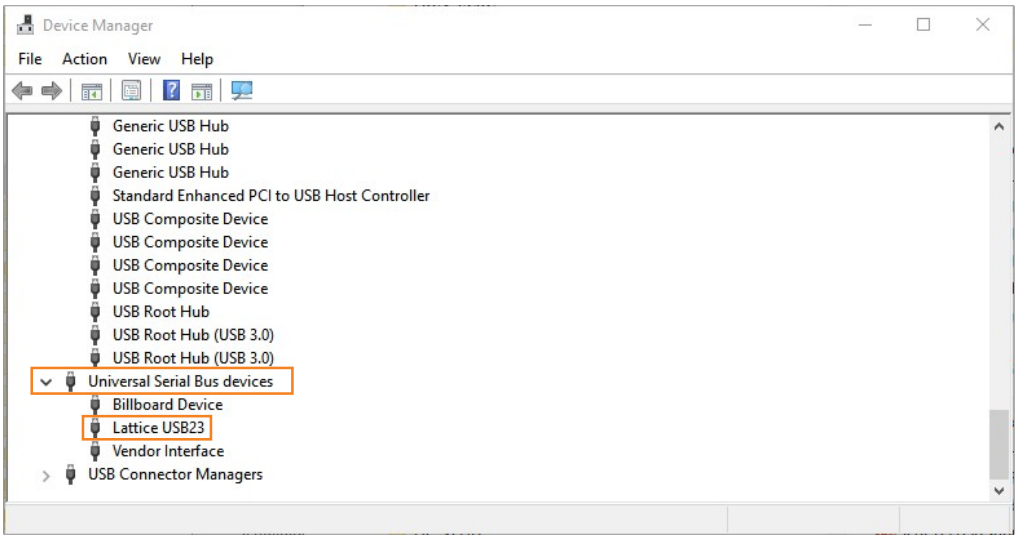

Figure 7.1. LIFCL-33U USB with Default WINUSB Driver in Windows Device Manager

In order to support Python script for testing and demo, the libusb-win32 driver is required. To install libusb-win32 Driver for USB23 on the LIFCL-33U Evaluation Board, perform the following steps:

1. Download the USB driver installation software from the Zadig web page. Zadig is a Windows application that installs generic USB drivers to help you access your USB devices.
2. Launch the downloaded *zadig-2.8.exe*. It requires the PC administration privilege to execute. The USB Driver Installation GUI will pop up as shown in Figure 7.2.

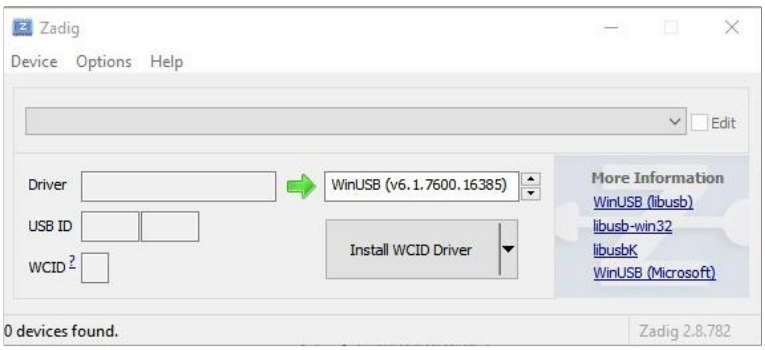

Figure 7.2. USB Driver Installation GUI

3. Click on **Options** > **List All Devices**.

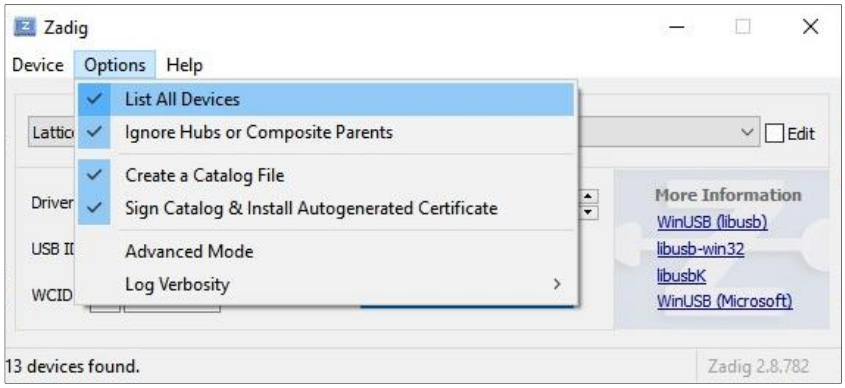

Figure 7.3. List All Devices in GUI

4. Select **Lattice USB23** Device from the pull-down menu. Double check the USB ID setting, then click the **Install/Reinstall Driver** button to install the USB driver.

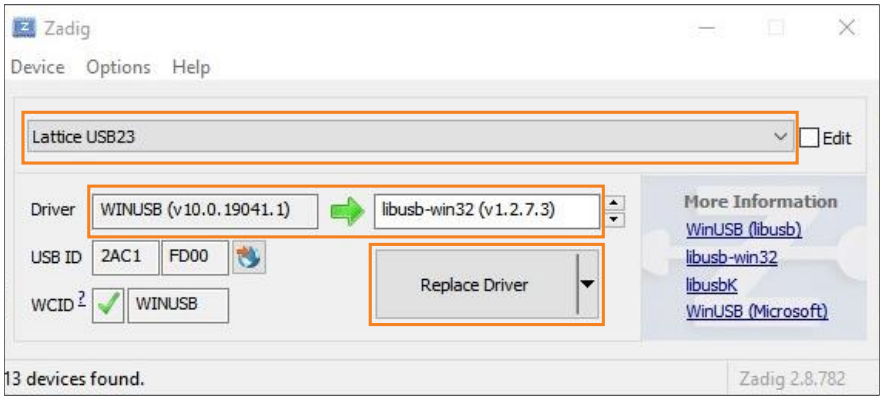

Figure 7.4. Install Driver for the Device

After the driver installation process, the USB on the LIFCL-33U will be shown under libsub-win32 devices in Windows Device Manager, as shown in Figure 7.5.

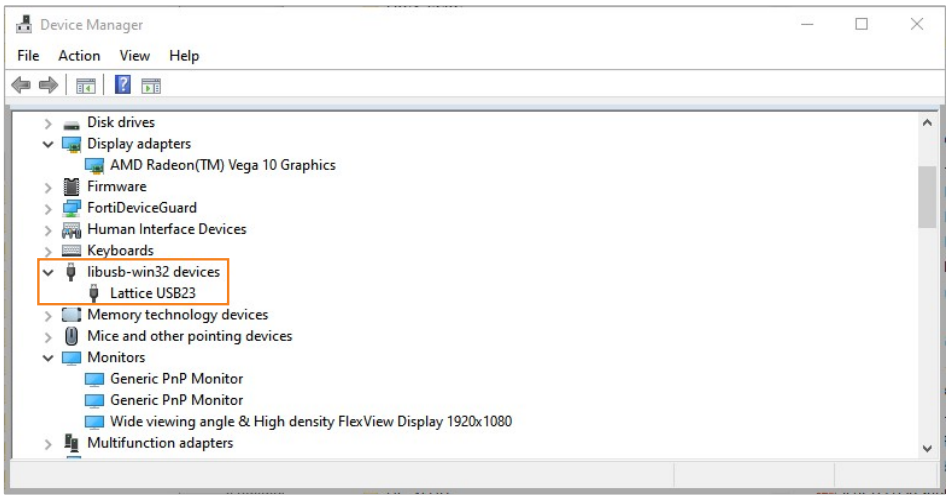

Figure 7.5. LIFCL-33U USB with libusb-win32 Driver in Windows Device Manager

7.3. I/O Aggregation Over USB Demonstration

The I/O Aggregation Over USB reference design can be demonstrated on the LIFCL-33U Evaluation Board for the following testing:

- GPIO input testing through a GPIO on Pin 7 of PMOD J1 on the I/O Daughter Board.
- GPIO output testing excise through LED0 (D5).
- I2C Controller testing through Pin 3 and P4 of PMOD J1 on the I/O Daughter Board to an EEPROM Memory Module with I2C interface.
- SPI Controller testing by accessing the on board SPI Flash memory (U5).

7.3.1. LIFCL-33U Evaluation Board Connection and Setup

The Pre-Programmed LIFCL-33U Evaluation Board should be connected as shown in Figure 7.6.

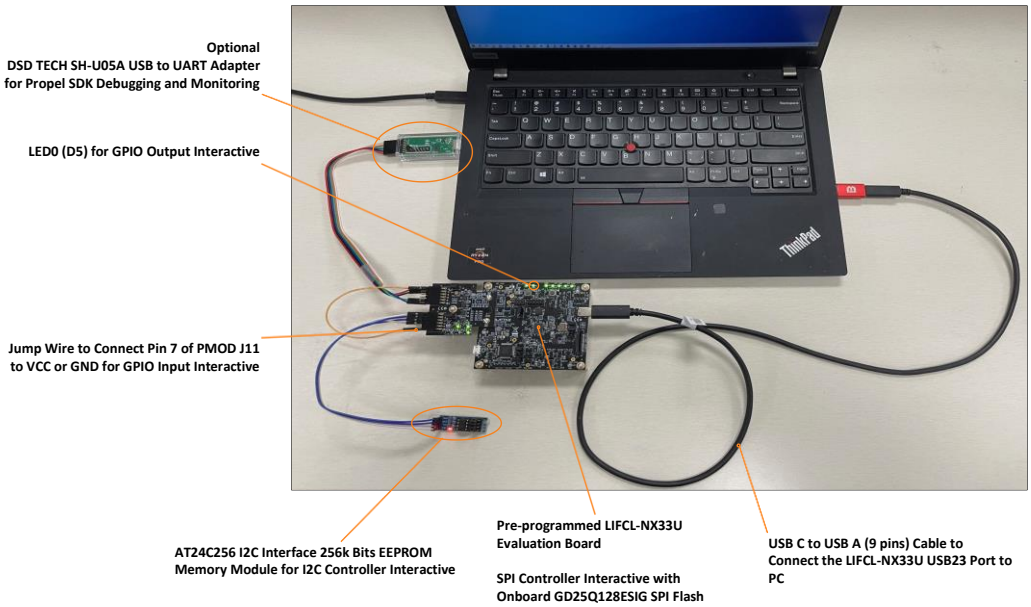

Figure 7.6. LIFCL-33U Evaluation Board Setup for Demonstration

SPI controller is directly connected to the on board SPI Flash. The connection table for I2C controller to EEPROM Memory Module, UART to UART adapter, and the GPIO input are shown in Table 7.1.

Table 7.1. Board Connection Table

LIFCL-33U Evaluation Board				Connection To	
Peripheral IP	**Function**	**I/O-DB PMOD**	**Pin Number**	**Pin**	**Device**
I2C Controller (Firmware Index 0)	VCC	J1	6 or 12	VCC	AT24C256 I2C Interface 256k Bits EEPROM Memory Module
	GND		5 or 11	GND	
	SDA		4	SDA	
	SCL		3	SCL	
UART	VCC	J2	6 or 12	VCC	DSD TECH SH-U05A USB to UART Adapter
	GND		5 or 11	GND	
	TXD		7	RX	
	RXD		8	TX	
GPIO (Firmware Index 0)	GPIO Input	J1	7	12 11	PMOD J1 Pin 12 for input 1 PMOD J1 Pin 11 for input 0

Make sure Jumper 3 and J4 on the I/O Daughter Board are installed in order to get VCC power supply on PMOD J1 and J2.

7.3.2. Utilizing UART Channel for Transaction Monitoring

If desired, you can switch to **Lattice Propel Software Build Tool** with I/O Aggregation Over USB Reference SDK project opened for Eclipse. To open serial terminal, click **Terminal** tab and click on **Open a terminal** icon highlighted in Figure 7.7.

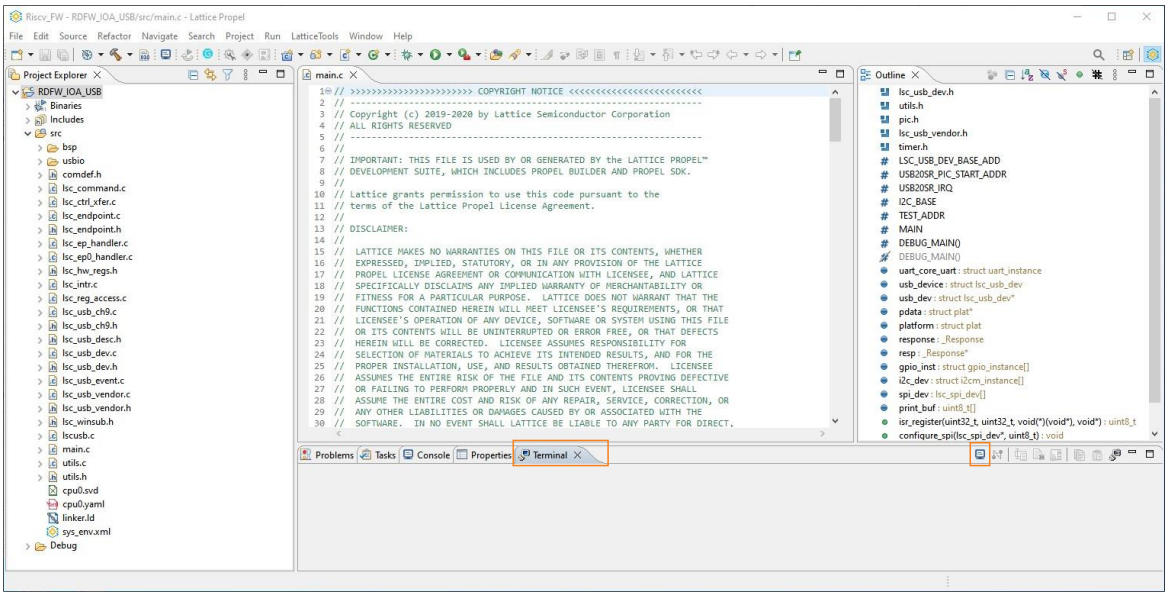

Figure 7.7. Open Serial Terminal

In the pop-up **Launch Terminal** window, select **Serial Terminal** from the drop down menu of **Choose Terminal**, as shown in Figure 7.8. Under **Settings**, select **Serial Port** corresponding to USB to UART convertor and select **Baud rate 115200** from the drop down menu as shown below.

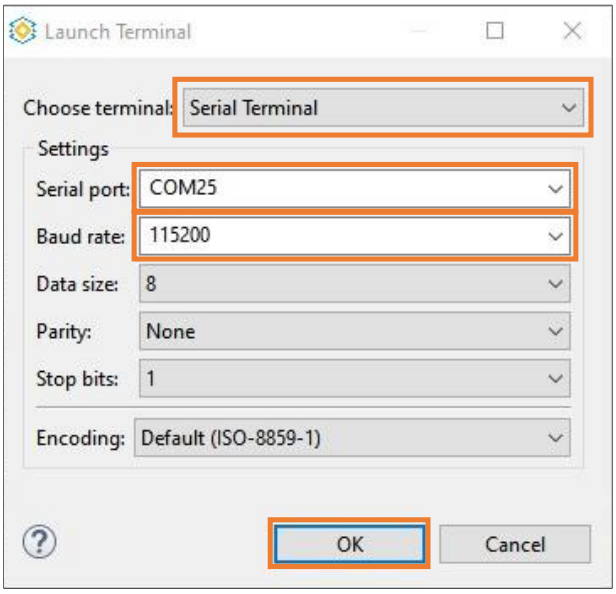

Figure 7.8. Select Serial COM Port

Under console window, the log message for PC command execution is similar to the example shown in Figure 7.9.

```
Problems  Tasks  Console  Properties  Terminal
COM25
 wLength =       0x01
 ~~~~~~~~~~ SPI IN Req ~~~~~~~~~
Inside spi isr
Interrupt Status 0x89
 SPI IP Read Data:  0x00,
 spi_op_ongoing = 0x00
 Vendor OUT Req
 EP0 STATE: 1
Setup Phase
 Vendor OUT Req
 EP0 STATE: 2
Data Parsing Phase
 <---- H2D
 bRequestType =  0x40
 bRequest =      002
 wValue =        000
 wIndex =        000
 wLength =       009
Received data from Host:  0x04  0x00  0x0a  0x00  0x03  0xfe  0x00  0x00  0x00

 SPI Request
  bRequestType =  0x40
  bRequest =      0x02
  wValue =        0x00
  wIndex =        0x00
  wLength =       0x09
 Request Buffer:  0x4  0x0  0xa  0x0  0x3  0xfe  0x0  0x0  0x0

spi_tx_len = 0x04 spi_rx_len = 00a
-- SPI TX Buffer:  [0] =  0x3   [1] =  0xfe   [2] =  0x0   [3] =  0x0
 SPI Transfer request
SPI READ or WRITE-READ CASE
 Vendor IN Req
----> D2H
 bRequestType =  0xc0
 bRequest =      0x02
 wValue =        0x00
 wIndex =        0x00
 wLength =       0x0a
 ~~~~~~~~~~ SPI IN Req ~~~~~~~~~
Inside spi isr
Interrupt Status 0x89
 SPI IP Read Data:  0x08,   0x07,   0x06,   0x05,   0x04,   0x03,   0x02,   0x01,   0x0a,   0x0b,
 spi_op_ongoing = 0x00
```

Figure 7.9. Log Message through Serial Terminal

7.3.3. Running Python Script

To excise the I/O Aggregation Over USB reference design on the LIFCL-33U Evaluation Board, a Python script is provided at *./test/RD_IOA_USB.py*. Executing this Python script without arguments will sequentially perform the activities listed below, which is shown in Figure 7.10.

- I2C Write
- I2C Read
- GPIO Write to blink the LED0 (D5) ON and OFF, twice.
- SPI Write
- SPI Read

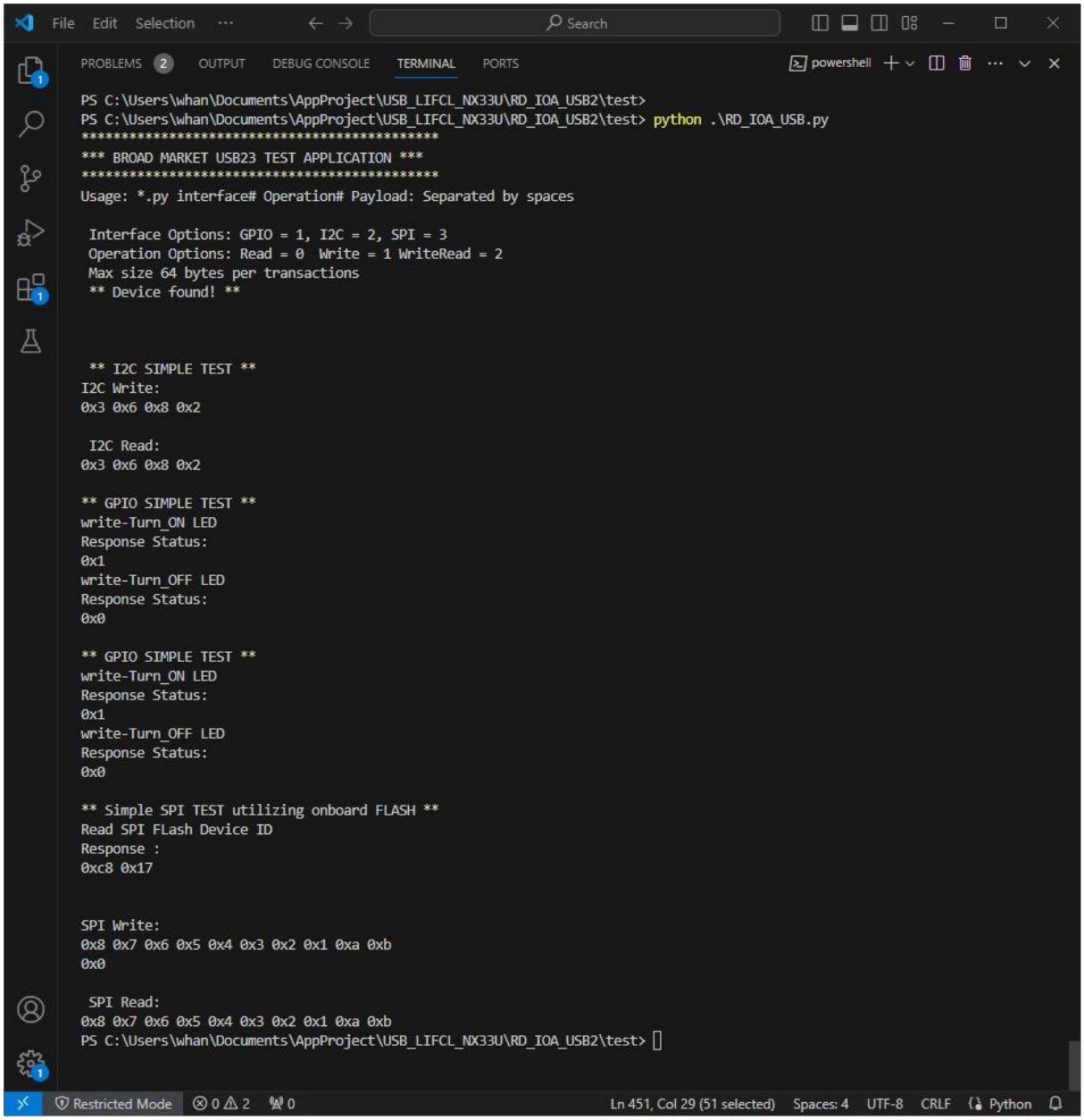

Figure 7.10. Python Script without Argument

To perform an individual activity, arguments have to be provided with the Python script execution.

The command line format is as follows:

```
python .\RD_IOA_USB.py [PERIPHERAL_SELECTION] [OPERATION_SELECTION] [PAY_LOAD]
```

Where,

PERIPHERAL_SELECTION	1	For GPIO
	2	For I2C Controller
	3	For SPI Controller
OPERATION_SELECTION	0	For Read
	1	For Write
	2	For WriteRead

For GPIO operations, there are only two I/Os defined for the reference design, one output to drive LED0 (D5) on Evaluation Board, and one input from Pin 7 of PMOD J11.

The PAY_LOAD values for different operations are as follows:

- The PAY_LOAD for GPIO operation is 1 byte in length, in which the bit 0 represent the output pin and bit 1 represent the input pin.
- For I2C operations, the PAY_LOAD should be formatted as:

  (1 byte Target Address + 2 byte TX length + 2 byte RX length + 59 byte I2C DATA)
  Note: The I2C DATA for AT24C256 I2C EEPROM Module should be (2byte EEPROM Offset + 57bytes Data).
- For SPI operation, the PAY_LOAD should be formatted as:

  (2 byte TX length + 2 byte RX Length + 60 bytes SPI DATA)
  Note: The SPI DATA for GD25Q128ESIG SPI Flash should be (1byte Command + 3byte Operand + 56bytes Data).

The following subsections provide examples of the Python script execution with arguments for individual activity. The command arguments are formatted in **orange** and the expected outcomes are formatted in **green**.

7.3.3.1. GPIO Operation Examples

- GPIO Write to turn LED0 on:

```
python .\RD_IOA_USB.py 1 1 1
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  1
Operation =  1
Payload =  [1]
Handling GPIO
GPIO Write: 0x1

 WR O.K:
```

- GPIO Write to turn LED0 off:

```
python .\RD_IOA_USB.py 1 1 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  1
Operation =  1
Payload =  [0]
Handling GPIO
GPIO Write: 0x0

 WR O.K:
```

- GPIO Read to sense the input pin state:

```
python .\RD_IOA_USB.py 1 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  1
Operation =  0
Payload =  [0]
Handling GPIO
GPIO Status: 0x2
```

7.3.3.2. I2C Operation Examples

- I2C operation to set Memory Module address pointer to zero:

```
python .\RD_IOA_USB.py 2 1 80 2 0 0 0 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  2
Operation =  1
Payload =  [80, 2, 0, 0, 0, 0, 0]
I2C Write: 0x50 0x2 0x0 0x0 0x0 0x0 0x0

 WR O.K:
```

- I2C operation to write two bytes data:

```
python .\RD_IOA_USB.py 2 1 80 4 0 0 0 0 0 1 2
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  2
Operation =  1
Payload =  [80, 4, 0, 0, 0, 0, 0, 1, 2]
I2C Write: 0x50 0x4 0x0 0x0 0x0 0x0 0x0 0x1 0x2
```

```
 WR O.K:
```

- I2C operation to go to offset and read 2 bytes:

```
python .\RD_IOA_USB.py 2 1 80 2 0 2 0 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  2
Operation =  1
Payload =  [80, 2, 0, 2, 0, 0, 0]
I2C Write: 0x50 0x2 0x0 0x2 0x0 0x0 0x0

 WR O.K:
```

- I2C operation to send read command to fetch the result:

```
python .\RD_IOA_USB.py 2 0 80 0 0 2 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **


interface =  2
Operation =  0
Payload =  [80, 0, 0, 2, 0]
Read bytes =  2

 I2C Read: 0x1 0x2
```

7.3.3.3. SPI Operation Examples

- SPI operation to read SPI FLASH ID:

```
python .\RD_IOA_USB.py 3 2 4 0 2 0 144 0 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions
 ** Device found! **

interface =  3
```

```
Operation =  2
Payload =  [4, 0, 2, 0, 144, 0, 0, 0]
SPI Write-Read: 0x4 0x0 0x2 0x0 0x90 0x0 0x0 0x0

 WR O.K:
 SPI Read: 0xc8 0x17
```

- SPI operation to set SPI FLASH WRITE ENABLE:

```
python .\RD_IOA_USB.py 3 1 1 0 0 0 6
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Max size 64 bytes per transactions
 ** Device found! **

interface =  3
Operation =  1
Payload =  [1, 0, 0, 0, 6]
SPI Write: 0x1 0x0 0x0 0x0 0x6

 WR O.K:
```

- SPI operation to check SPI FLASH WIP flag:

```
python .\RD_IOA_USB.py 3 2 1 0 1 0 5
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions

interface =  3
Operation =  2
Payload =  [1, 0, 1, 0, 5]
SPI Write-Read: 0x1 0x0 0x1 0x0 0x5

 WR O.K:

 SPI Read: 0x2 (WEN = 1, WIP = 0)
```

- SPI operation to perform SPI FLASH sector erase:

```
python .\RD_IOA_USB.py 3 1 4 0 0 0 32 254 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Max size 64 bytes per transactions
 ** Device found! **
```

```
interface =  3
Operation =  1
Payload =  [4, 0, 0, 0, 32, 254, 0, 0]
SPI Write: 0x4 0x0 0x0 0x0 0x20 0xfe 0x0 0x0

 WR O.K:
```

- SPI operation to check SPI FLASH WIP flag:

```
python .\RD_IOA_USB.py 3 2 1 0 1 0 5
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions


interface =  3
Operation =  2
Payload =  [1, 0, 1, 0, 5]
SPI Write-Read: 0x1 0x0 0x1 0x0 0x5

 WR O.K:

 SPI Read: 0x0 (WEN = 0, WIP = 0)
```

- SPI operation to set SPI FLASH WRITE ENABLE:

```
python .\RD_IOA_USB.py 3 1 1 0 0 0 6
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Max size 64 bytes per transactions
 ** Device found! **

interface =  3
Operation =  1
Payload =  [1, 0, 0, 0, 6]
SPI Write: 0x1 0x0 0x0 0x0 0x6

 WR O.K:
```

- SPI operation to write 10 bytes data:

```
python .\RD_IOA_USB.py 3 1 14 0 0 0 2 254 0 0 8 7 6 5 4 3 2 1 10 11
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Max size 64 bytes per transactions
 ** Device found! **
```

```
interface =  3
Operation =  1
Payload =  [14, 0, 0, 0, 2, 254, 0, 0, 8, 7, 6, 5, 4, 3, 2, 1, 10, 11]
SPI Write: 0xe 0x0 0x0 0x0 0x2 0xfe 0x0 0x0 0x8 0x7 0x6 0x5 0x4 0x3 0x2 0x1 0xa 0xb

 WR O.K:
```

- SPI operation to check SPI FLASH WIP flag:

```
python .\RD_IOA_USB.py 3 2 1 0 1 0 5
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions

interface =  3
Operation =  2
Payload =  [1, 0, 1, 0, 5]
SPI Write-Read: 0x1 0x0 0x1 0x0 0x5

 WR O.K:

 SPI Read: 0x0 (WEN = 0, WIP = 0)
```

- SPI operation to read 10 bytes data back:

```
python .\RD_IOA_USB.py 3 2 4 0 10 0 3 254 0 0
*********************************************
*** BROAD MARKET USB23 TEST APPLICATION ***
*********************************************
Usage: *.py interface# Operation# Payload: Separated by spaces

 Interface Options: GPIO = 1, I2C = 2, SPI = 3
 Operation Options: Read = 0  Write = 1 WriteRead = 2
 Max size 64 bytes per transactions

interface =  3
Operation =  2
Payload =  [4, 0, 10, 0, 3, 254, 0, 0]
SPI Write-Read: 0x4 0x0 0xa 0x0 0x3 0xfe 0x0 0x0

 WR O.K:

 SPI Read: 0x8 0x7 0x6 0x5 0x4 0x3 0x2 0x1 0xa 0xb
```

8. Customizing the Reference Design

To customize or modify the I/O Aggregation Over USB reference design by adding or removing any peripheral soft IP module requires a RISC-V firmware update. Contact Lattice Sales for more information.

Appendix A. Resource Utilization

Table A.1 shows the resource utilization of the I/O Aggregation over USB Reference Design for LIFCL-33U-8CTG104C using Synplify Pro of Lattice Radiant software 2024.2.

Table A.1. Resource Utilization for LIFCL-33U-8CTG104C

Module/Resource Utilization		LUTs	Registers	EBRs	Large RAMs
Peripherals	GPIO	46	31	0	0
	I2C Controller	775	567	0	0
	SPI Controller	483	406	2	0
Microcontroller System	RISC-V + sysMEM + Bus controller and converters	8,915	4,470	5	2
Total		10,219	5,474	7	2
Percentage Over the Device		37%	20%	11%	40%

References

- Lattice Radiant Software User Guide
- Lattice Propel SDK User Guide (FPGA-UG-02195)
- Lattice Propel Builder User Guide (FPGA-UG-02196)
- RISC-V MC CPU IP User Guide (FPGA-IPUG-02210)
- AHB-Lite Interconnect Module User Guide (FPGA-IPUG-02051)
- AHB-Lite to APB Bridge Module User Guide (FPGA-IPUG-02053)
- System Memory Module User Guide (FPGA-IPUG-02073)
- GPIO IP Core User Guide (FPGA-IPUG-02076)
- SPI Controller IP Core User Guide (FPGA-IPUG-02069)
- SPI Target IP Core User Guide (FPGA-IPUG-02070)
- I2C Controller IP User Guide (FPGA-IPUG-02071)
- I2C Target IP User Guide (FPGA-IPUG-02072)
- UART 16550 IP User Guide (FPGA-IPUG-02100)
- CrossLink-NX web page
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- Lattice Insights web page for Lattice Semiconductor training courses 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 2.0, December 2024

Section	Change Summary
All	Made editorial fixes.
Abbreviations in This Document	Added *I/O*, *I/O-DB*, and *PC*.
Introduction	Updated the *Software Tool and Version*, *IP Version*, *Board*, and *Cable* fields in Table 1.1. Summary of the Reference Design.
Signal Description	In Table 4.1. Primary I/O: • Added the *I/O-DB Access* column. • Updated the information for the *gpio_0_z[1]*, *i2cm_scl*, *i2cm_sda*, *spim_sclk_o*, *spim_ssn_o*, *spim_mosi_o*, *spim_miso_i*, *uart_txd_o*, and *uart_rxd_i* parameters.
Building the Reference Design	Removed the Lattice Propel software version in the Running Propel SDK Project, Opening Propel SDK Project, Running Propel Builder Design, and Opening Propel Builder Design sections.
LIFCL-33U Evaluation Board Programming	• Updated the introductory paragraph in the LIFCL-33U Evaluation Board Programming section. • Updated Figure 6.1. Radiant Programmer GUI.
Running the Reference Design on Evaluation Board	• Updated the LIFCL-33U Evaluation Board Connection to PC section. • Added *I/O Daughter Board* to the I/O Aggregation Over USB Demonstration section. • Updated Figure 7.6. LIFCL-33U Evaluation Board Setup for Demonstration. • Replaced the *PMOD* column with *I/O DB PMOD* in Table 7.1. Board Connection Table. • Removed the Lattice Propel software version in the introductory paragraph of the Utilizing UART Channel for Transaction Monitoring section.
Resource Utilization	• Moved the *Resource Utilization* to *Appendix A*. • Updated the resource utilization for the Lattice Radiant software version 2024.2.
References	Updated the *Lattice Radiant Software User Guide*, *Lattice Propel SDK User Guide (FPGA-UG-02195)*, and *Lattice Propel Builder User Guide (FPGA-UG-02196)*.

Revision 1.1, October 2024

Section	Change Summary
Introduction	Updated the *Simulation/Validation* information in Table 1.1. Summary of the Reference Design.

Revision 1.0, May 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com