



ELBERT S7 HANDBOOK

An Educational Guide to FPGA Design and
Development

COPYRIGHT and PROPRIETARY

Table of Contents

SECTION I	4
1. INTRODUCTION	4
2. BOARD FEATURES	6
2.1. FPGA Device	6
2.2. Configuration and Memory	6
2.3. Communication Interface	7
2.4. User I/O	7
2.5. Peripheral Interfaces	7
3. APPLICATION	8
4. WIRING DIAGRAM	9
5. USB Interfacing and Programming Options	10
6. JTAG Connector	11
7. PROG_B and Reset Buttons	11
8. Power Supply	12
9. USER I/O	13
9.1. LEDs	13
9.2. DIP Switches	13
9.3. PMOD Headers	14
10. PERIPHERAL INTERFACES	16
10.1. Seven Segment Display	16
10.2. HDMI TRANSMITTER	17
10.3. Micro SD Card	18
10.4. Gigabit Ethernet	19
10.5. Audio Jack	20
11. Generating Bitstream Using Vivado	21
12. Programming Elbert S7 Using JTAG	22
13. Programming Elbert S7 Using USB-JTAG	24
14. Programming QSPI Flash using Vivado.	25

SECTION II.....	27
TRAIN ON THE BOARD	27
Getting Started with Vivado: Creating a New FPGA Project	27
1. UART Communication – Printing “Hello World”	30
2. Controlling Onboard Peripherals (LEDs, Seven Segment Display, and PMOD) with Slide Switches	43
3. HDMI OUTPUT EXAMPLE DESIGN	49
4. DDR3 Mem test on ELBERT S7	57
5. Gigabit Ethernet Example Design	71
6. SD card test	88
REFERENCES	104



The board is an ideal choice for anyone interested in **digital design, embedded systems, signal processing, or rapid prototyping**. With strong support for industry-standard tools like **Vivado** and **Vitis**, and compatibility with both **Verilog** and **VHDL**, the Elbert S7 provides a smooth and scalable learning path for anyone stepping into the field of FPGA design.



2. BOARD FEATURES

The Elbert S7 FPGA development board offers a rich set of features, making it a powerful tool for learning and developing digital systems. Its well-balanced combination of core components and peripherals allows users to implement a wide variety of applications—from basic logic design to advanced embedded systems.

2.1. FPGA Device

- **Model:** AMD Spartan-7 (XC7S50)
- **Package:** CSGA324
- **Speed Grade:** -1

This low-power, high-performance FPGA offers enough logic resources and I/O options for both educational and practical digital design tasks.

2.2. Configuration and Memory

- **DDR3 SDRAM:** MT41J128M16JT-125: KTR, A high-speed 2 Gb (128M x 16) DDR3 memory chip used for applications requiring larger memory capacity, such as video processing, buffering, or running soft processors.
- **Flash Memory:** 128 Mb Quad-SPI (MT25QU128ABA1ESE-oSIT TR)
Used to store FPGA configuration bitstreams and other application data.
- **Clock Source:** 100 MHz CMOS oscillator
Serves as the main clock input to drive system timing.
- **Configuration Methods:**
 - JTAG (standard FPGA programming method)
 - USB (via onboard FTDI chip; supported on Both Windows and Linux systems)

2.3. Communication Interface

- **FTDI FT2232H:** Dual-channel USB-to-serial/FIFO interface
Enables high-speed communication between the FPGA and a host PC. Also supports USB-based FPGA programming and debugging.

2.4. User I/O

- **8 DIP Switches & 8 LEDs:**
Useful for creating simple input/output projects and learning digital logic control.
- **Four PMOD Headers:**
Standard 2x6 connectors for attaching external modules and user-defined peripherals.

2.5. Peripheral Interfaces

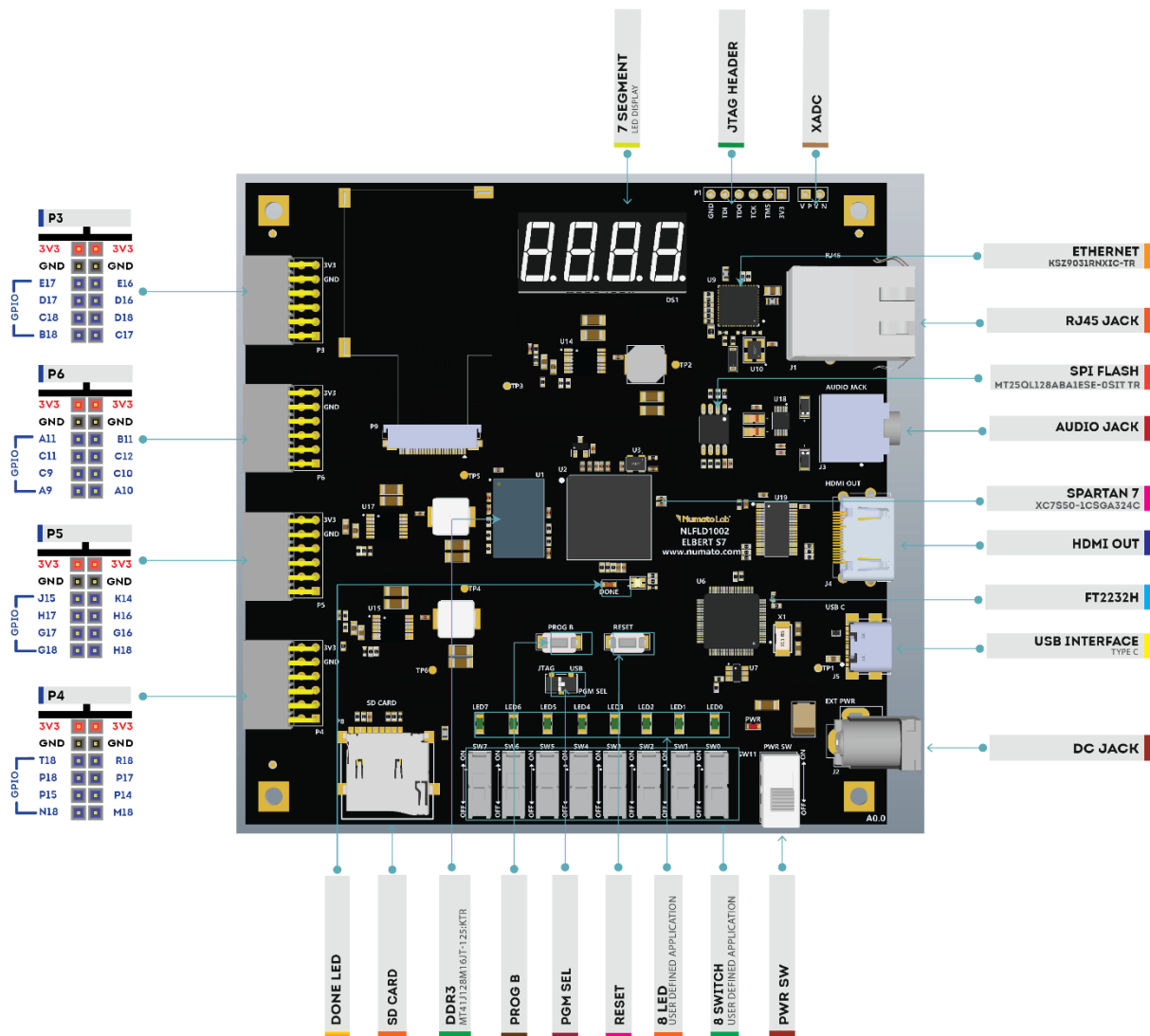
- **SD Card Slot:**
Provides external storage, useful for embedded applications requiring file access.
- **HDMI Transmitter:**
Enables the board to send video signals to external displays, suitable for image processing or display generation projects.
- **Gigabit Ethernet Port:**
Supports high-speed networking, ideal for IoT, remote monitoring, or network-based applications.
- **Audio Jack:**
Allows audio input/output for sound-related projects.
- **Seven-Segment Display:**
Displays numeric or limited alphanumeric information, ideal for counter or output visualization tasks.

3. APPLICATION

The Elbert S7 is a versatile development board suitable for a wide range of educational and practical applications:

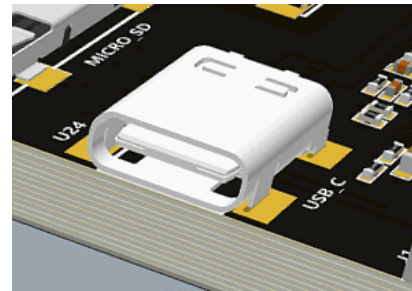
- **Educational Use:** Ideal for teaching digital design and embedded systems in schools and universities.
- **Prototype Development:** Enables rapid hardware prototyping and testing of new product ideas.
- **Accelerated Computing:** Supports hardware-based acceleration of compute-intensive tasks.
- **Custom Processor Design:** Allows development and testing of soft-core or custom embedded processors.
- **Signal Processing:** Suitable for implementing and testing real-time digital signal processing applications.
- **Communication Systems:** Enables the design and evaluation of communication protocols and devices.
- **Video Processing:** Supports projects involving HDMI output and image/video processing tasks.

4. WIRING DIAGRAM



5. USB Interfacing and Programming Options

The Elbert S7 board features a high-speed USB interface powered by the FTDI FT232H chip, enabling seamless communication with Windows, Linux, or macOS computers. A standard USB Type-A to Type-C cable is used to connect the board to a host system. The USB connection also supplies power to the board by default, so it's important to avoid connecting it to overloaded or unpowered USB hubs.



To provide flexibility in programming, the board includes a programming mode selection mechanism. A multiplexer (MUX) is used internally to switch between two configuration sources:

- JTAG (via external programmer)
- USB-JTAG (via onboard FT232H chip)

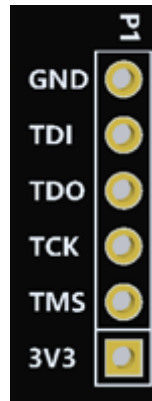
The FT232H's Channel A is configured specifically for USB-based FPGA programming. A PGM SEL switch is provided on the board, allowing users to select the desired programming method:

- Set the switch to USB to use onboard USB-JTAG.
- Set the switch to JTAG to program via an external JTAG programmer.



This flexible configuration makes the Elbert S7 suitable for both beginners using USB for simplicity and advanced users who prefer direct JTAG access.

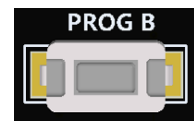
6. JTAG Connector



The Elbert S7 board includes a standard JTAG header that provides access to the FPGA's internal JTAG registers. This interface supports programming and debugging using tools such as the Xilinx Platform Cable USB. Users can connect a compatible JTAG cable to this header to perform direct configuration, in-system debugging, or low-level hardware testing. This option is especially useful for advanced users requiring greater control and visibility into the FPGA during development.

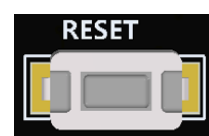
7. PROG_B and Reset Buttons

Elbert S7 features a Push-button **PROG_B** normally meant to be used as a “PROG_B” signal for configuration reset. Push-button **PROG_B** is connected to FPGA pin **R8**. For enabling manual configuration reset, push-button **PROG_B** is connected to GND. The user can reconfigure the FPGA manually, by pressing this push-button **PROG_B**.



“PROG_B” controls the configuration logic. When the PROG_B pin is de-asserted, resets the FPGA and initializes the new configuration.

Elbert S7 features a Push-button **RESET** normally meant to be used as “Reset” signal for designs running on FPGA. Push-button **RESET** is connected to FPGA pin **T14**. Push-button **RESET** is **active-high**. This push button can also be used for any other input and is not just limited to be used as a Reset signal.



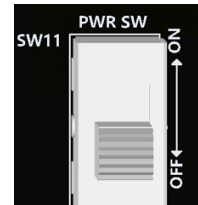
8. Power Supply

The **Elbert S7** board can be powered in two ways: using a **USB Type-C** cable or a **5V DC adapter**. You can choose whichever is more convenient.

The board is designed to automatically select the best power source if both are connected, so you don't need to worry about switching manually.

It also includes built-in protection and filtering to ensure the board always receives clean and stable power, making it reliable for both development and real-world use.

A user-accessible **PWR SW** (Sw11) is provided to manually control the power delivery to the board. This switch adds convenience by allowing users to turn the board ON or OFF without needing to disconnect the power cable physically.



Finally, an onboard Power LED (**PWR**) connected after the switch provides a visual indication when the board is powered.



Note: Only a regulated **5V DC power supply** should be used to power the board through either the USB Type-C port or the DC jack. Supplying a voltage higher than 5V may damage the internal circuitry.

9. USER I/O

9.1. LEDs

The board features 8 user-controllable LEDs. These LEDs serve as visual indicators and are commonly used for debugging or representing binary output data. Each LED is connected to a dedicated FPGA pin and can be driven directly using logic outputs.

PIN NAME	FPGA PIN	IO STANDARD
LED 0	V14	LVCOMS33
LED 1	V15	LVCOMS33
LED 2	U12	LVCOMS33
LED 3	V13	LVCOMS33
LED 4	T12	LVCOMS33
LED 5	T13	LVCOMS33
LED 6	R11	LVCOMS33
LED 7	T11	LVCOMS33

Table 1: LED Pin Mapping

9.2. DIP Switches

There are 8 onboard DIP switches that can be used to input binary data or control signals into the FPGA. These inputs are ideal for controlling the flow of a design, setting modes, or triggering actions within a design.

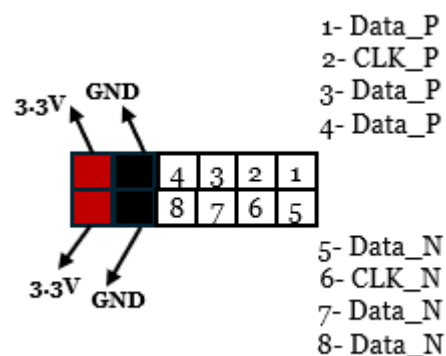
PIN NAME	FPGA PIN	IO STANDARD
SW 0	C4	LVCOMS18
SW 1	B4	LVCOMS18
SW 2	C3	LVCOMS18
SW 3	B3	LVCOMS18
SW 4	A5	LVCOMS18
SW 5	A4	LVCOMS18
SW 6	A3	LVCOMS18
SW 7	A2	LVCOMS18

Table 2: DIP Switch Pin Mapping

9.3. PMOD Headers

PMOD (Peripheral Module) connectors are a widely adopted standard for connecting peripheral devices to FPGA and microcontroller development boards. The Elbert S7 development board provides **four 2×6 PMOD headers** (P3, P4, P5, P6), allowing users to interface a variety of external modules including sensors, displays, communication interfaces, and custom digital circuits.

Each PMOD header offers **eight general-purpose I/O (GPIO) signals** plus **two dedicated power pins (3.3V and GND)**, organized in a dual-row format. The I/O lines from each PMOD are connected directly to the FPGA, enabling flexible software-defined control and configuration.



PIN NAME	FPGA PINS				IO STANDARD
	CONNo(P1)	CONN1(P2)	CONN2(P3)	CONN3(P4)	
D0	M18	H18	A10	C17	LVC MOS33
D1	P14	G16	C10	D18	LVC MOS33
D2	P17	H16	C12	D16	LVC MOS33
D3	R18	K14	B11	E16	LVC MOS33
D4	N18	G18	A9	B18	LVC MOS33
D5	P15	G17	C9	C18	LVC MOS33
D6	P18	H17	C11	D17	LVC MOS33
D7	T18	J15	A11	E17	LVC MOS33

Table 2: PMOD Pin Mapping



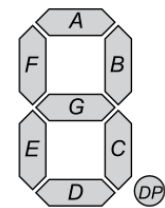
10. PERIPHERAL INTERFACES

10.1. Seven Segment Display

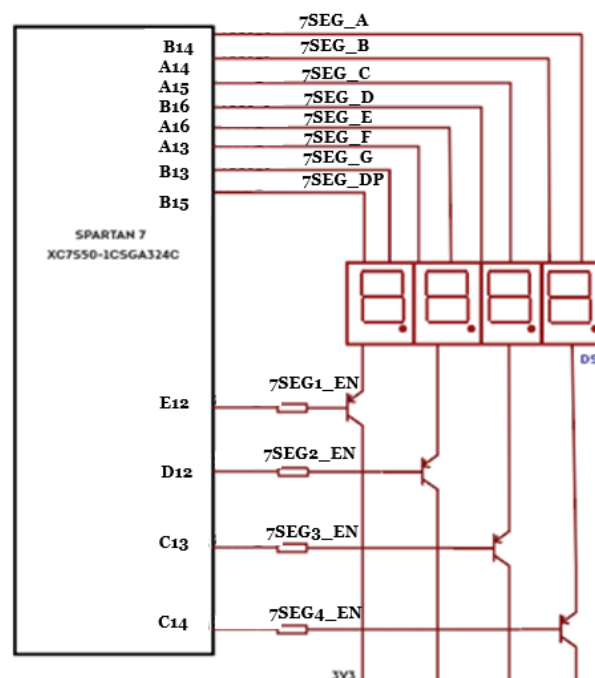
A seven-segment display is a simple output device commonly used for displaying decimal numbers or characters. The **Elbert S7** includes a 4-digit seven segment display, which is useful for displaying counters, timers, or debugging information.

Each digit is made up of 7 individual segments (labelled A–G) and an optional decimal point (dp). The display works by quickly switching between digits using control signals—a technique known as multiplexing.

You can control which digit is shown by activating the digit's enable line and then lighting up the correct segments to form a number or letter.



Note: All signals (*a, b, c, d, e, f, g, dot, enable 1, enable 2, enable 3, enable 4*) used for controlling 7-Segment display are **active-low** signals. So, for example, for displaying “8” in display-2, users need to drive *Enable 2* to 0 as well as drive signals *a, b, c, d, e, f* to 0. All other signals need to be driven to 1.



10.2. HDMI TRANSMITTER

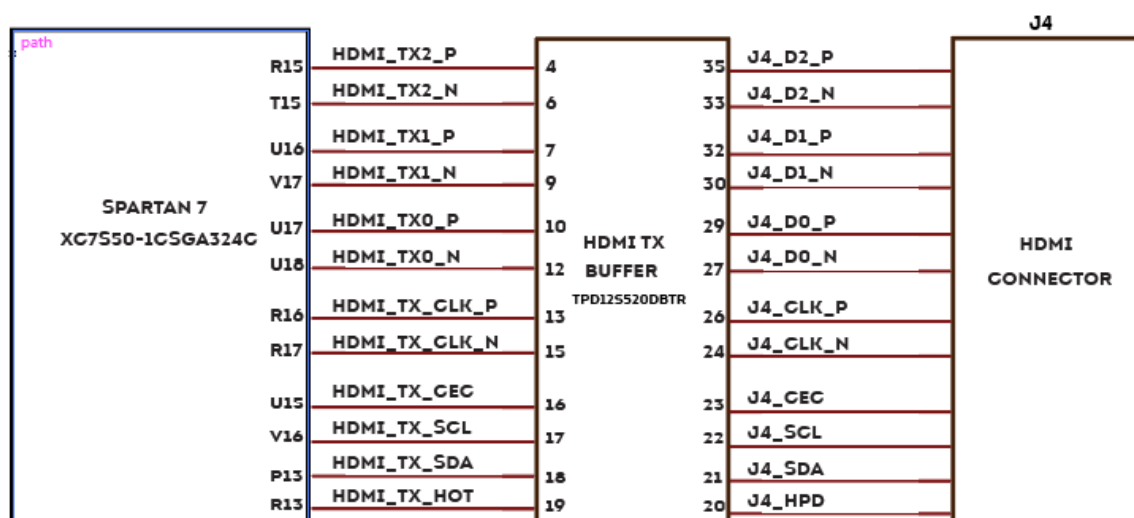
The HDMI (High-Definition Multimedia Interface) Transmitter on the Elbert S7 board allows the FPGA to send digital video and audio signals to an external display such as a monitor or TV.

HDMI uses a high-speed signalling technology known as **TMDs (Transition-Minimized Differential Signalling)**. This ensures reliable transmission of data over HDMI cables by reducing electromagnetic interference and ensuring signal integrity.

Inside the FPGA, video signals are generated in digital format—usually in RGB (Red, Green, Blue) colour space—along with synchronization signals such as horizontal sync, vertical sync, and data enable. These signals are then encoded and sent through the HDMI transmitter chip to the display device.

The HDMI interface typically includes the following signals:

- **TMDs Data Channels (3 pairs):** Carry video/audio/control data.
- **TMDs Clock:** Synchronizes the data.
- **DDC (I2C lines):** Used to read the display's capabilities (EDID)
- **HPD:** Lets the FPGA know a display is connected.
- **CEC:** Optional control communication between HDMI devices



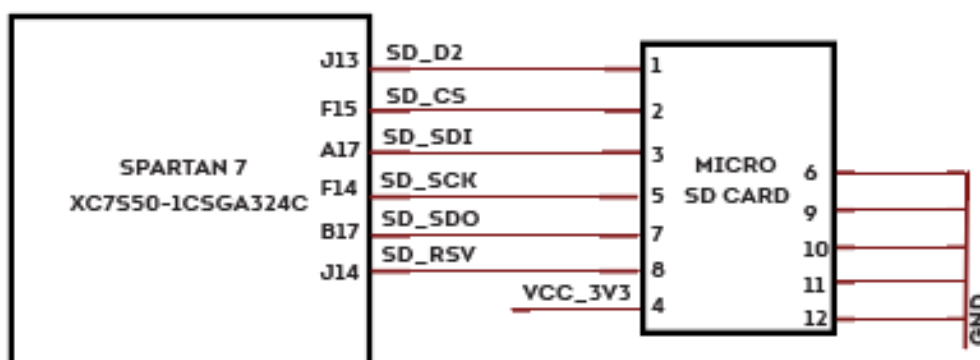
10.3. Micro SD Card

The Elbert S7 board features a **microSD card slot**, which allows users to interface with removable flash memory for data storage and retrieval. This is especially useful for embedded applications where storing files, logs, configuration settings, or multimedia content is required.

The SD card slot on the board is wired to the FPGA through an **SPI (Serial Peripheral Interface)**. SPI is a simple and widely used protocol for communicating with memory devices like SD cards. Although SD cards also support a more complex SD bus mode, SPI is preferred in FPGA designs due to its simplicity and ease of implementation.

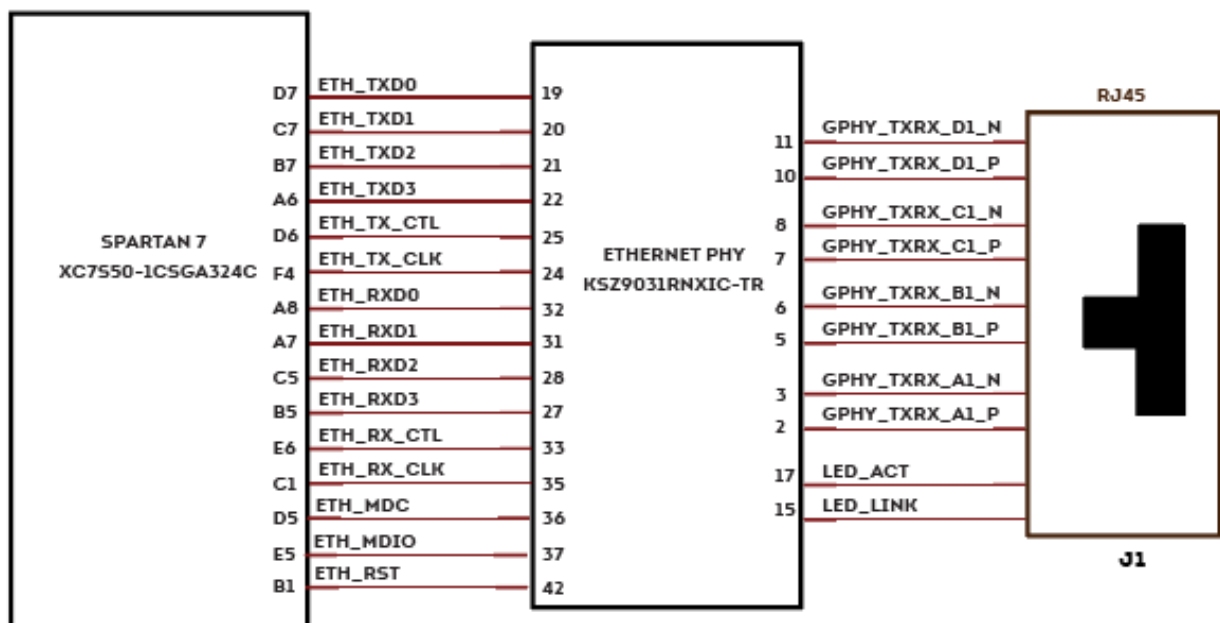
The typical SPI signals used are:

- **MOSI (Master Out Slave In)** – Data sent from the FPGA to the SD card.
- **MISO (Master In Slave Out)** – Data received from the SD card to the FPGA.
- **SCLK (Serial Clock)** – Clock signal generated by the FPGA to control communication timing.
- **CS (Chip Select)** – Used to select and activate the SD card.



10.4. Gigabit Ethernet

Elbert S7 Development Board features KSZ9031RNX, a highly integrated Ethernet transceiver from Microchip that comply with 10BASE-T, 100BASE-TX, and 1000Base-T IEEE 802.3 standards. It supports communication with the Ethernet MAC layer via standard RGMII interface. KSZ9031RNX implements auto-negotiation to automatically determine the best possible speed and mode of operation. It contains a high-performance 10/100/1000T transceiver and the RGMII interface supports 1000Mbps (1Gbps) operation.

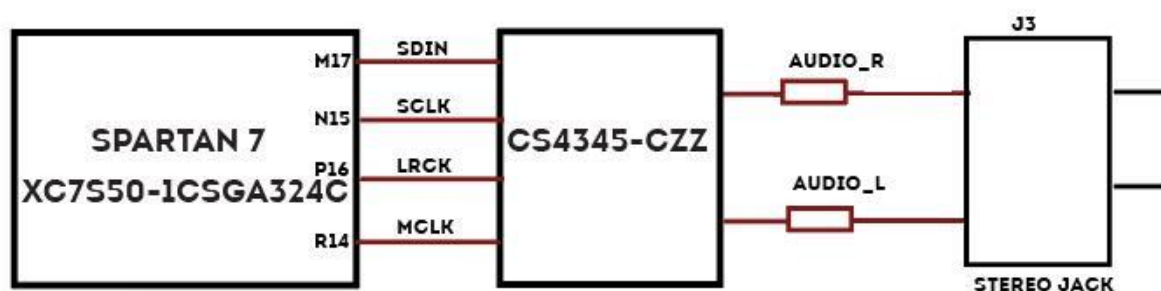


10.5. Audio Jack

The Elbert S7 FPGA board is equipped with a 3.5mm stereo audio jack, enabling audio output to external speakers or headphones. This output is managed by a dedicated digital-to-analog converter (DAC) — the CS4345-CZZ, a high-performance stereo DAC from Cirrus Logic. The inclusion of this audio interface makes the board well-suited for multimedia, audio signal processing, and embedded audio playback applications.

Key Features

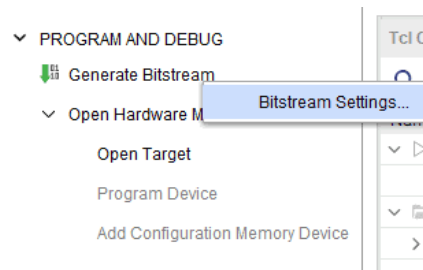
- **Stereo Output:** Supports two audio channels (Left and Right) for full stereo playback.
- **High-Quality DAC:** The CS4345 provides low distortion and high dynamic range, resulting in clear and high-fidelity sound.
- **Digital Audio Interface:** The DAC accepts audio data in **I²S (Inter-IC Sound)** format, a standard serial protocol for transmitting PCM audio between digital audio devices.
- **16/24-bit Audio Support:** Capable of processing CD-quality and high-resolution audio streams.
- **Headphone Compatible:** The output is designed to drive line-level audio, which can be connected to headphones or powered speakers.



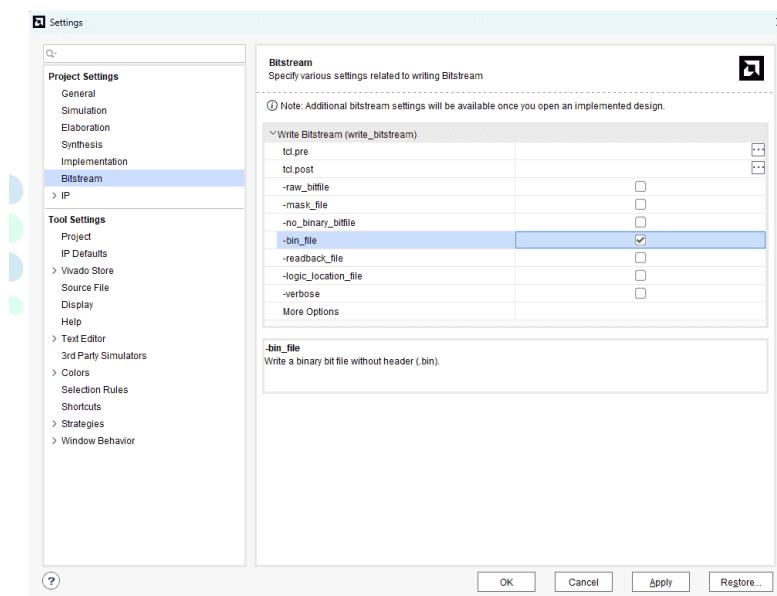
11. Generating Bitstream Using Vivado

The bitstream can be generated for Elbert S7 in Vivado by following the steps below:

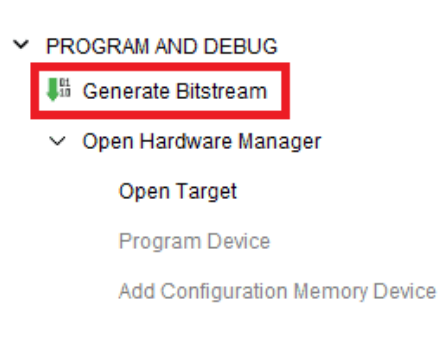
Step 1: It is recommended to generate .bin bitstream file along with .bit bitstream file. Click “Bitstream Settings”.



Step 2: Select “-bin_file*” option in the dialog window and Click OK.



Step 3: Finally click “Generate Bitstream”.



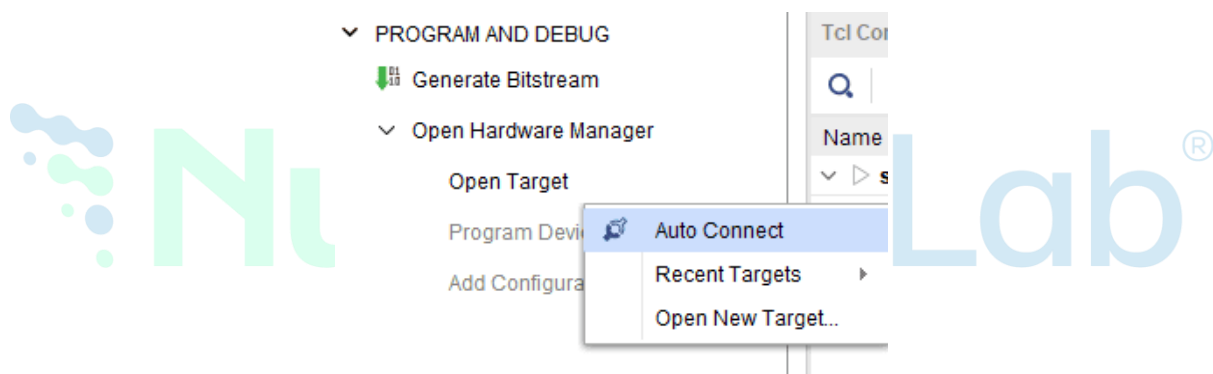
12. Programming Elbert S7 Using JTAG

Set Switch **PGM_SEL** to **JTAG** for JTAG programming.

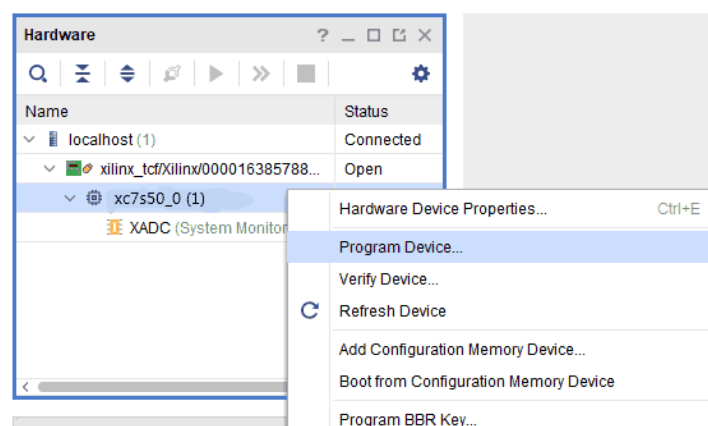
Elbert S7 FPGA features an onboard JTAG connector which facilitates easy reprogramming of SRAM and onboard SPI flash through JTAG programmer like “AMD Platform cable USB”. Following steps illustrate how to program FPGA on Elbert using JTAG.

Step 1: By using JTAG cable, connect AMD platform cable USB to Elbert S7 and power it up.

Step 2: Open Vivado project and open the target by clicking on the “Open Target” in “Open Hardware Manager” in the “Program and Debug” section of the Flow Navigator window. Select “Auto Connect”.

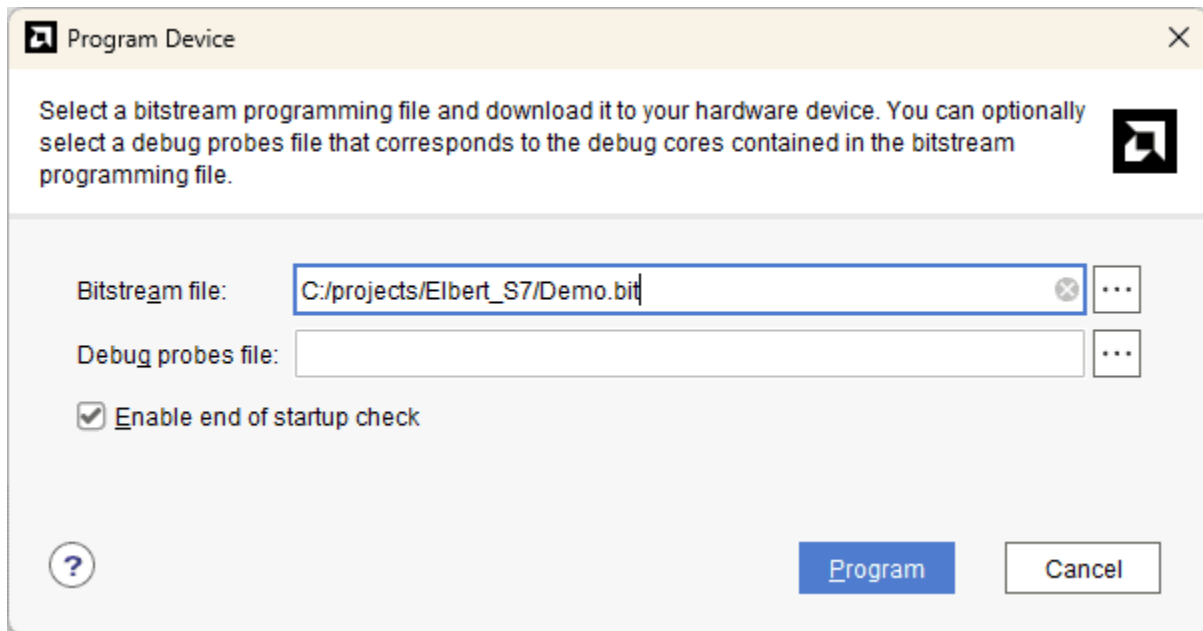


Step 3: If the device is detected successfully, then select “Program Device” after right clicking on the target device “XC7S50_0” as shown below.



Step 4: In the dialog window which opens, Vivado automatically chooses correct bitstream file if the design was synthesized, implemented and bitstream generated

successfully. If needed, browse to the bitstream which needs to be programmed to FPGA. Finally, click “Program”.

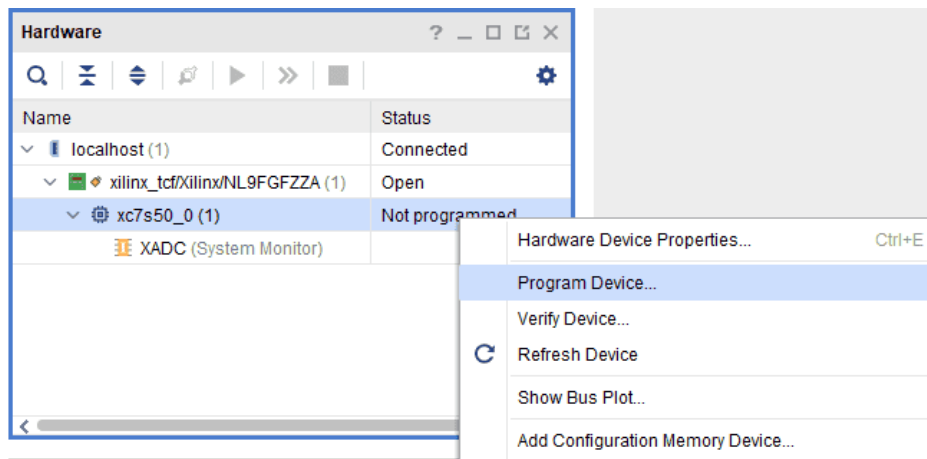


As soon as “Program” is clicked, a green coloured DONE LED (DONE) on Elbert S7 should light up, indicating that programming process is going on. This LED will turn off when the configuration is complete.

13. Programming Elbert S7 Using USB-JTAG

Ensure that the [D2XX drivers](#) are installed prior to programming. The channel A of FTDI FT2232H chip on Elbert S7 board is connected to the JTAG interface of the FPGA. Through this connection, USB interface can be used as a JTAG programmer, eliminating the need for a dedicated JTAG cable or connector. Following steps illustrate how to program FPGA on Elbert S7 using USB.

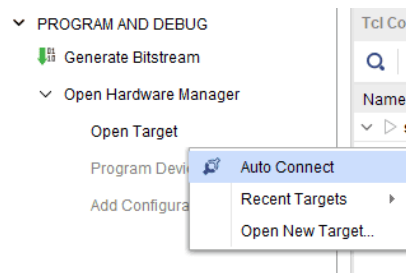
1. Ensure that Switch **PGM_SEL** is set to **USB** and Connect the USB Type-C cable to the FPGA board.
2. Click on “Auto connect” under hardware manager and it will automatically establish the connection.



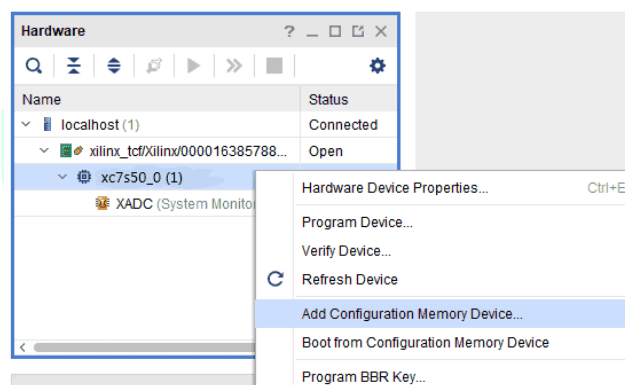
14. Programming QSPI Flash using Vivado.

A .bin or .mcs file is required for programming Elbert S7 onboard QSPI flash.

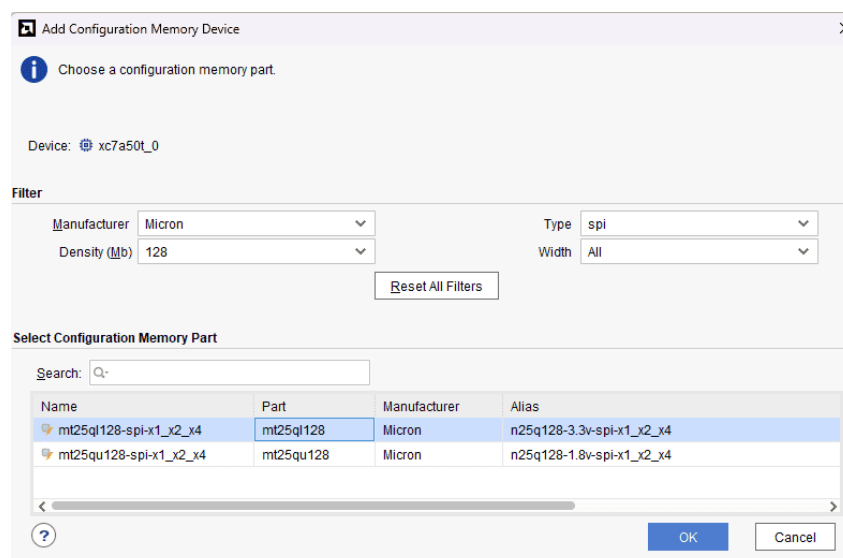
Step 1: Open Vivado project and open the target by clicking on the “Open Target” in “Open Hardware Manager” in the “Program and Debug” section of the Flow Navigator window. Select “Auto Connect”.



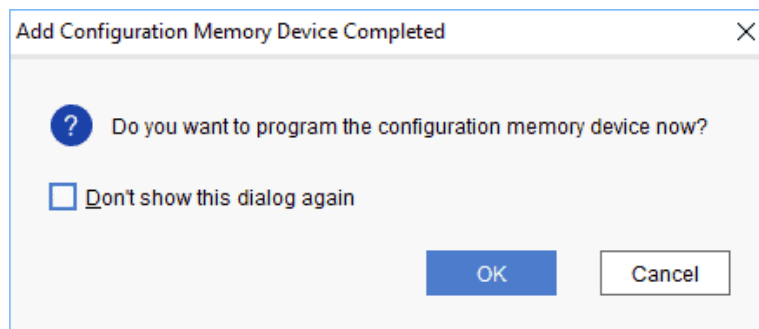
Step 2: If the device is detected successfully, then select “Add Configuration Memory Device” after right clicking on the target device “xc7s50_0” as shown below.



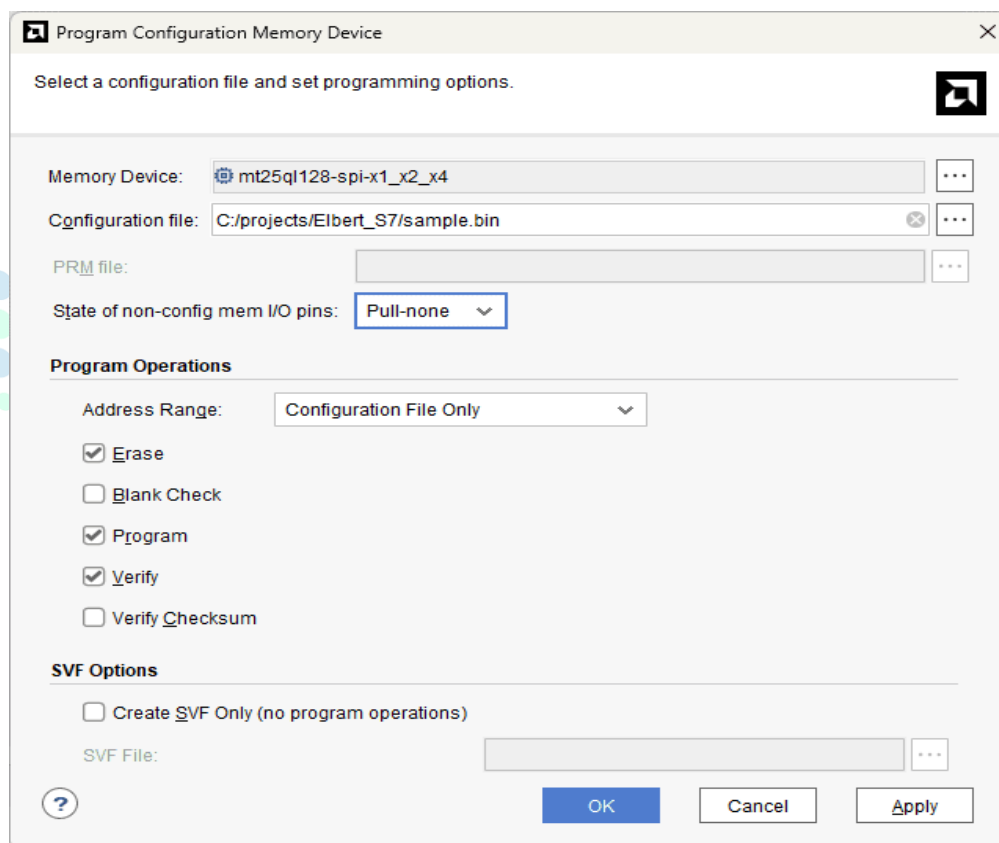
Step 3: Select the memory device “mt25ql128-spi-x1_x2_x4”, then click OK.



Step 4: After completion of Step 3 the following dialog box will open. Click OK.



Step 5: Browse to the working .bin file or the .mcs file (whichever applicable) and click OK to program as shown below. If programming is successful, a confirmation message will be displayed.



SECTION II

TRAIN ON THE BOARD

Getting Started with Vivado: Creating a New FPGA Project

Before diving into the peripheral interface projects, it is essential to understand the basic procedure for setting up a new project in Vivado tailored for the Elbert S7 FPGA board. This section will guide you through the initial steps, including downloading and configuring the Board Support Package (BSP).

To begin, download the **Elbert S7 BSP** from our official [GitHub](#) repository and place it in the appropriate board files directory (follow the readme file in GitHub repo) on your computer. This allows Vivado to recognize the Elbert S7 board during project creation, simplifying IP integration and pin assignments.

The procedures described in this part will remain consistent across all peripheral interface projects throughout **Section II**. By following this workflow, you'll ensure that your development environment is correctly set up, enabling a smooth and efficient design experience.

Prerequisites:

Hardware:

- Elbert S7 FPGA Development Board.
- Xilinx Platform Cable USB II JTAG debugger. (optional)
- USB A to USB Type C cable.
- 5V DC power supply.

Software:

- Vivado Design Suite with Vitis installed (2024.1)
- Serial terminal application (PuTTY, Tera Term, etc.)

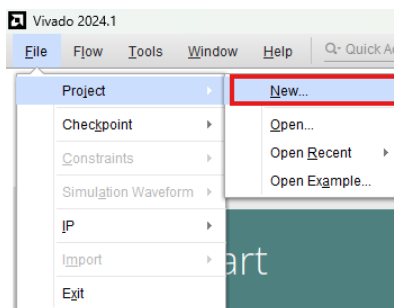
Basic procedures to create new project in Vivado.

Step 1:

Download and install the Vivado Board Support Package files for Elbert S7 from [here](#). Follow the README.md file on how to install Vivado board support files for Numato Lab boards.

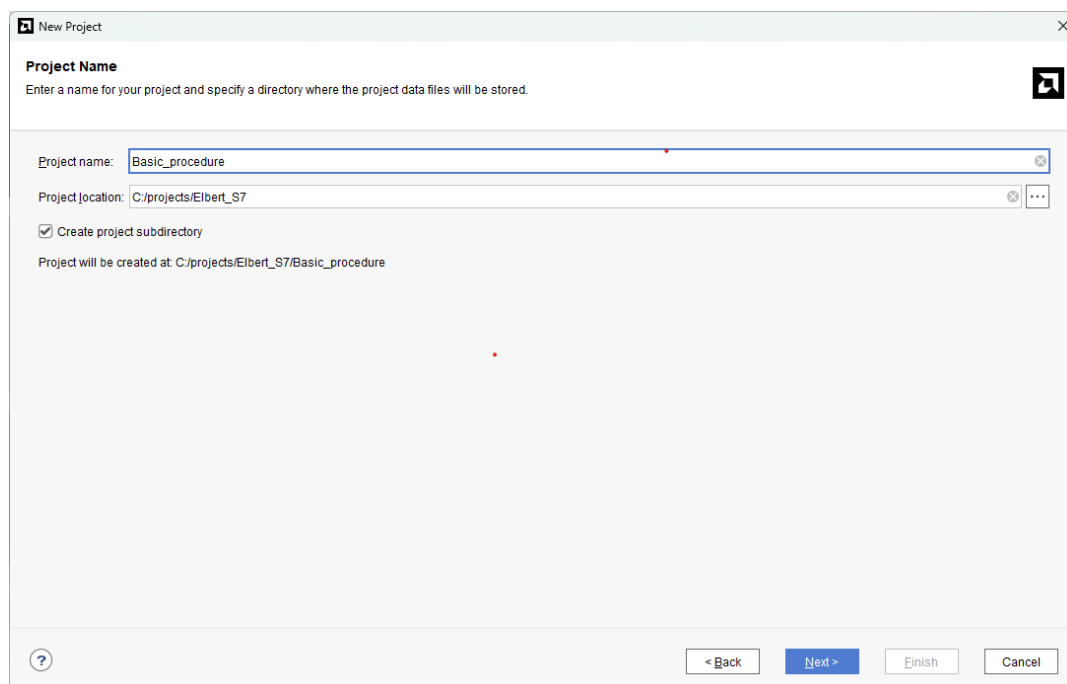
Step 2:

Open the AMD Vivado Design suite, go to “File -> Project -> New” to create a new project. The “New project” window will pop up. Click “Next”.

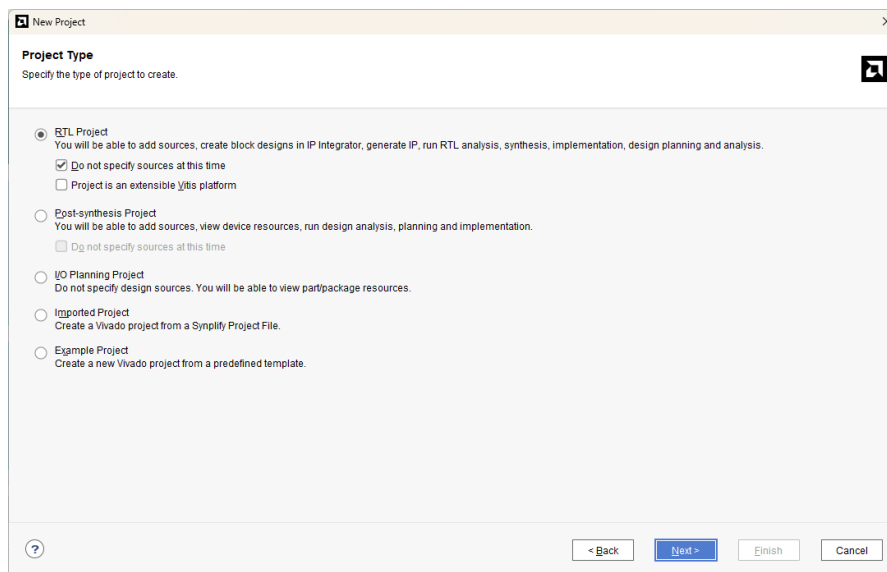


Step 3:

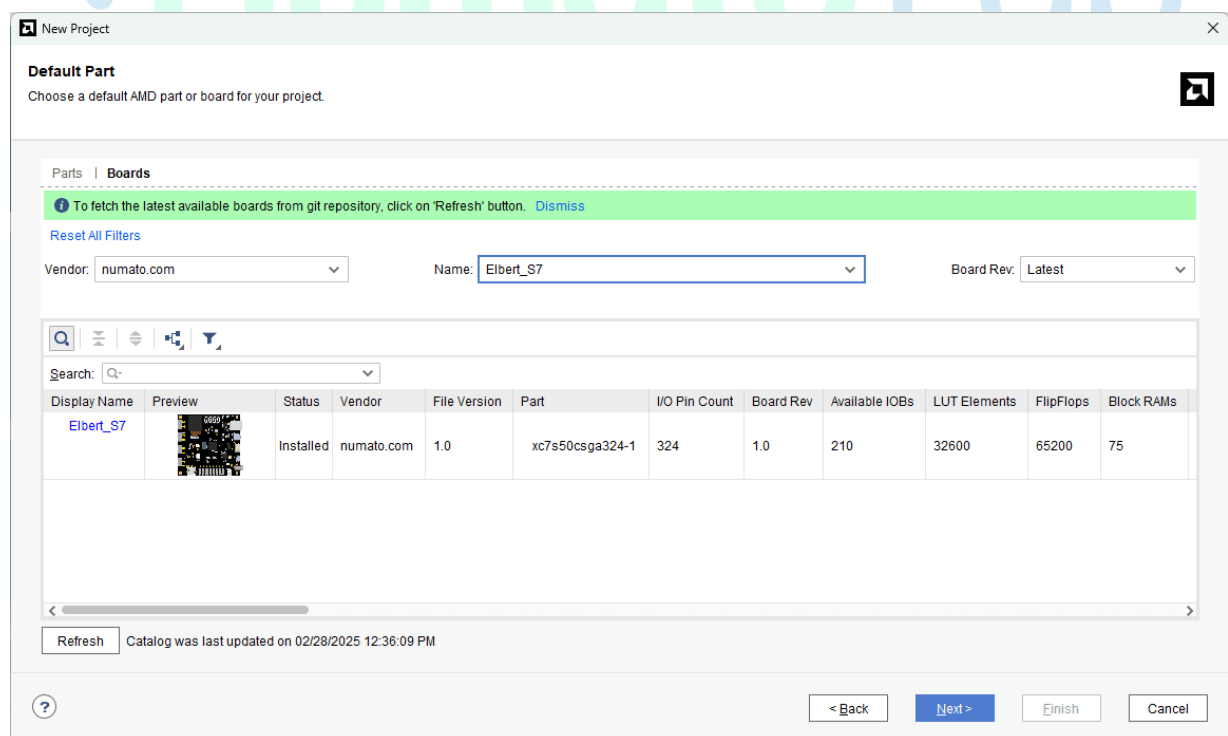
In the “Project Name” window, enter a name for the project and save it at a suitable location. Select the option “Create project subdirectory” to keep all the project files in a single folder.



Now you will see the “Project Type” page as shown below. Select the “RTL Project” and select the option “Do not specify sources at this time”. Click “Next”.



Step 4: In the “Default Part” window, select the “Boards” tab. Choose the Vendor as “numato.com”, filter the Name “Elbert_S7” and select the board as shown below. Click “Next” to continue. If Elbert S7 is not displayed in the boards list, make sure that the [board support files](#) are installed correctly.



In the next window, click “Finish” to complete creating the new project. When the new project wizard exits, a new project will be created by Vivado with the specified settings.

1. UART Communication – Printing “Hello World”

INTRODUCTION

In this first hands-on project, we will verify the UART (Universal Asynchronous Receiver/Transmitter) interface on the Elbert S7 FPGA board by printing a simple "Hello World" message to the serial terminal. UART is a widely used serial communication protocol that allows the FPGA to communicate with a host PC or other devices using simple text-based messaging.

This project serves as a basic sanity check to ensure the UART peripheral is functioning correctly and that the board can transmit data over a serial connection. It also helps familiarize users with integrating IP cores and observing output through a terminal emulator like Tera Term or PuTTY.

By completing this project, users will gain confidence in creating Vivado projects, generating bitstreams, using the BSP, and verifying output via UART communication.

Creating Microblaze based Hardware Platform for Elbert S7 ®

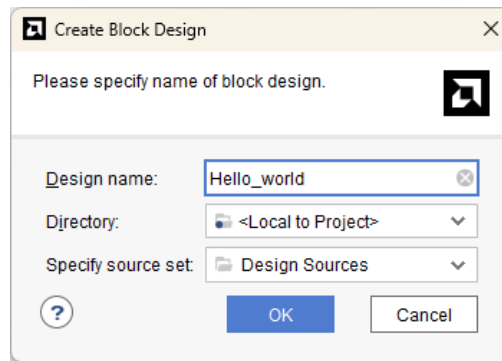
The following steps will walk you through the process of creating a new project with Vivado and building a hardware platform with MicroBlaze soft processor using an IP integrator.

STEP 1:

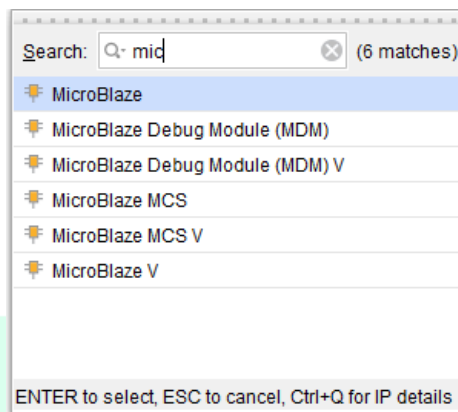
To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section “[Getting Started with Vivado: Creating a New FPGA Project.](#)”

STEP 2:

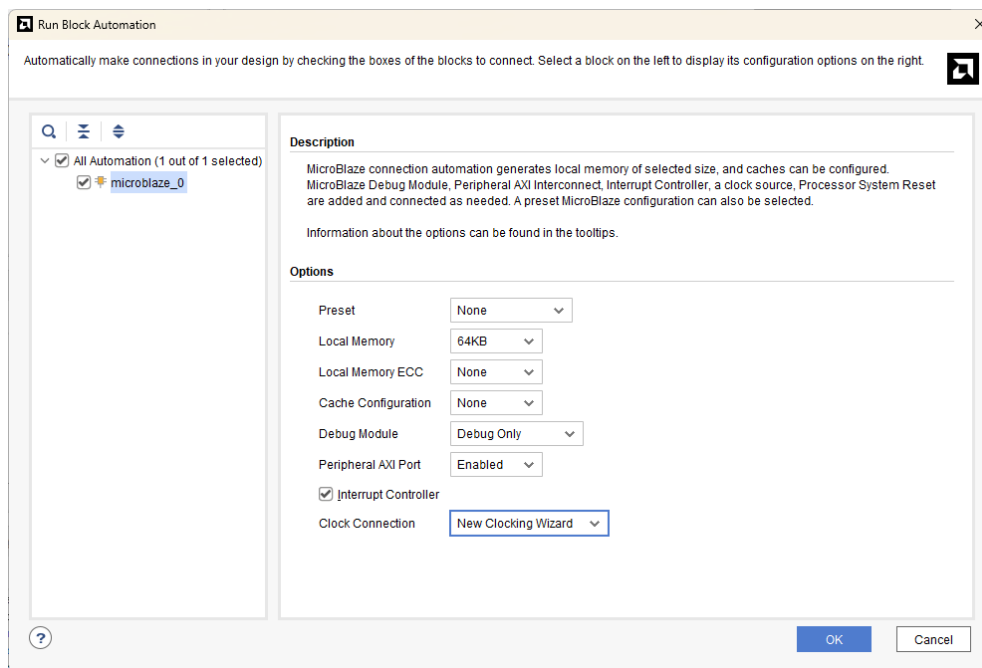
After creating a new project successfully, In the “Flow Navigator” panel, select “Create Block Design” under the IP integrator section. Give an appropriate name (Eg: “Hello_world”) to the design and click “OK”.

**Step 3:**

Go to Diagram window, right click and select “Add IP” from the popup menu. Search for “MicroBlaze” and add it to the design by double-clicking it.

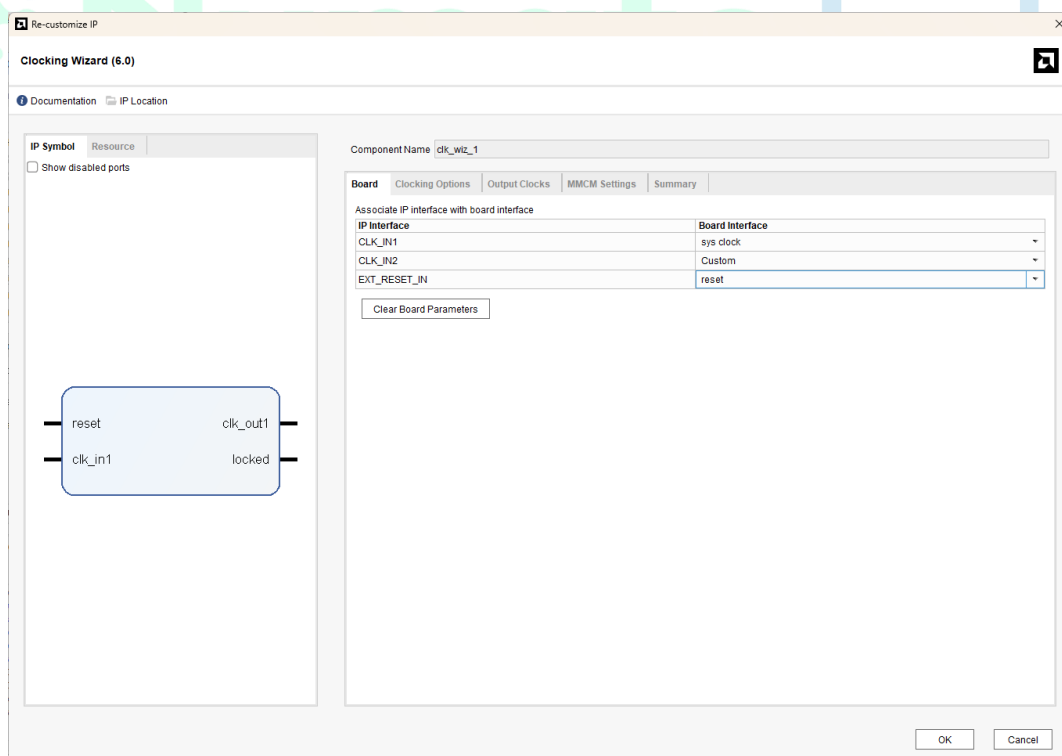


Click “Run Block Automation” present in the “Designer Assistance available” bar on the top left corner of the window to complete the design. Select the settings as shown in the following image. Click “OK” for Vivado to automatically configure the blocks for you.



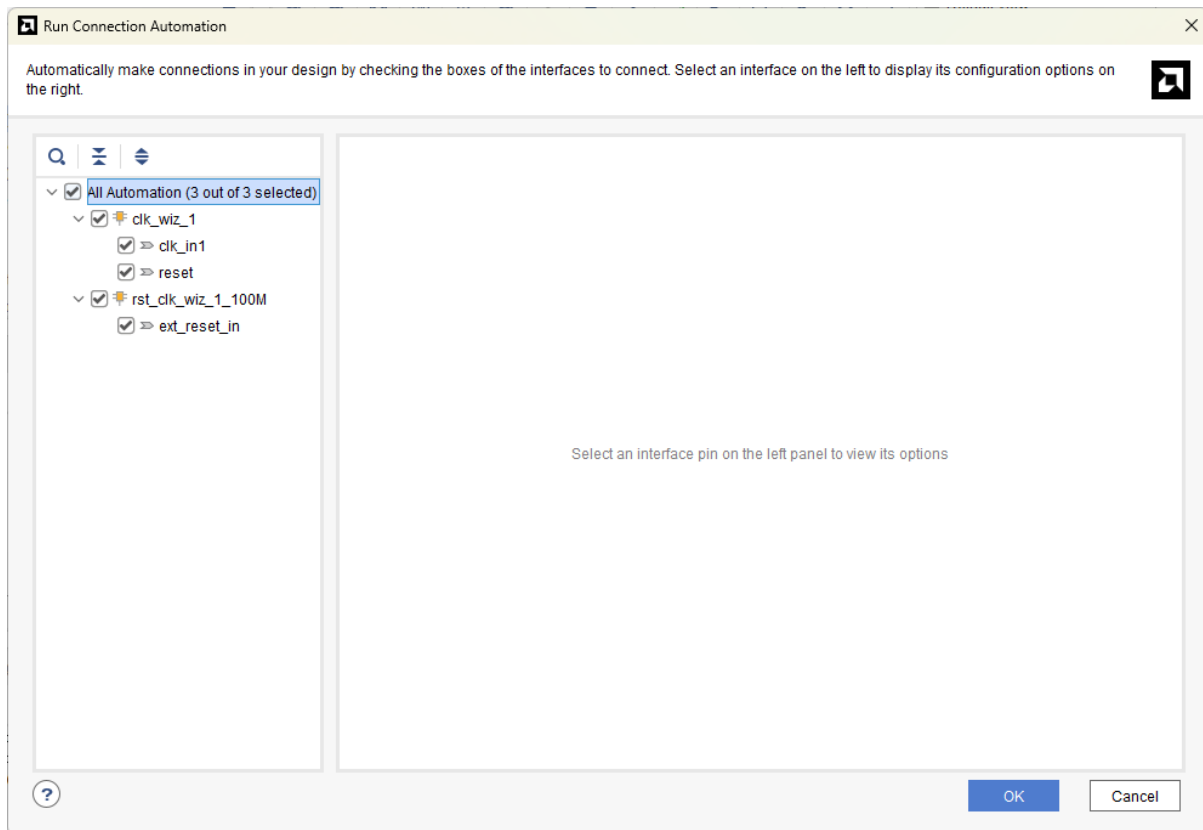
Step 4:

Double click “Clocking Wizard” IP and customize “Board” settings as shown in the following image.

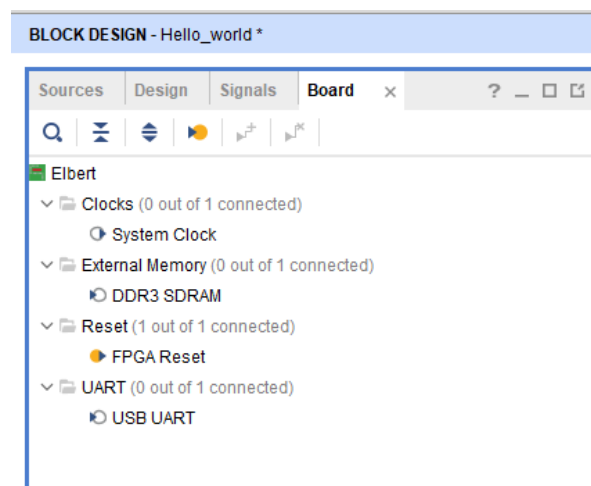


Step 5:

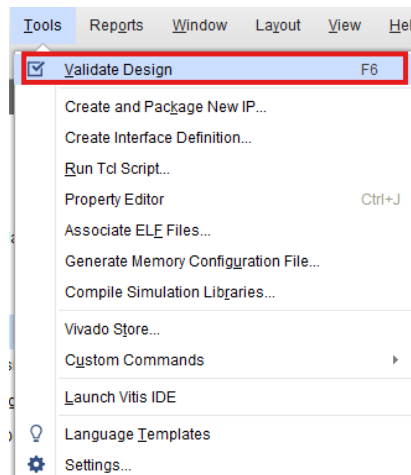
Run “Connection Automation” and select all the pins.

**Step 6:**

Go to the Board section, Drag and drop the USB UART from the Board section to the design.

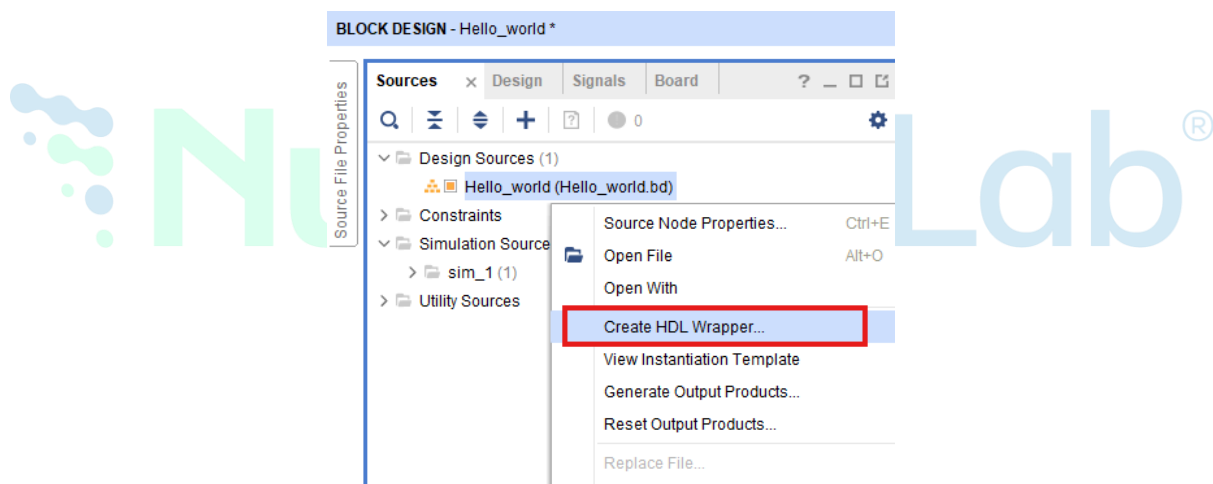


Click on “Run Connection Automation” select all the pins and click ok.



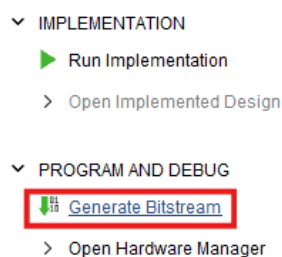
Step 9:

Right-click “Hello_world” in the “Sources” window and select “Create HDL Wrapper” from the popup menu. Click “OK” on the window that appears to finish generating a wrapper.



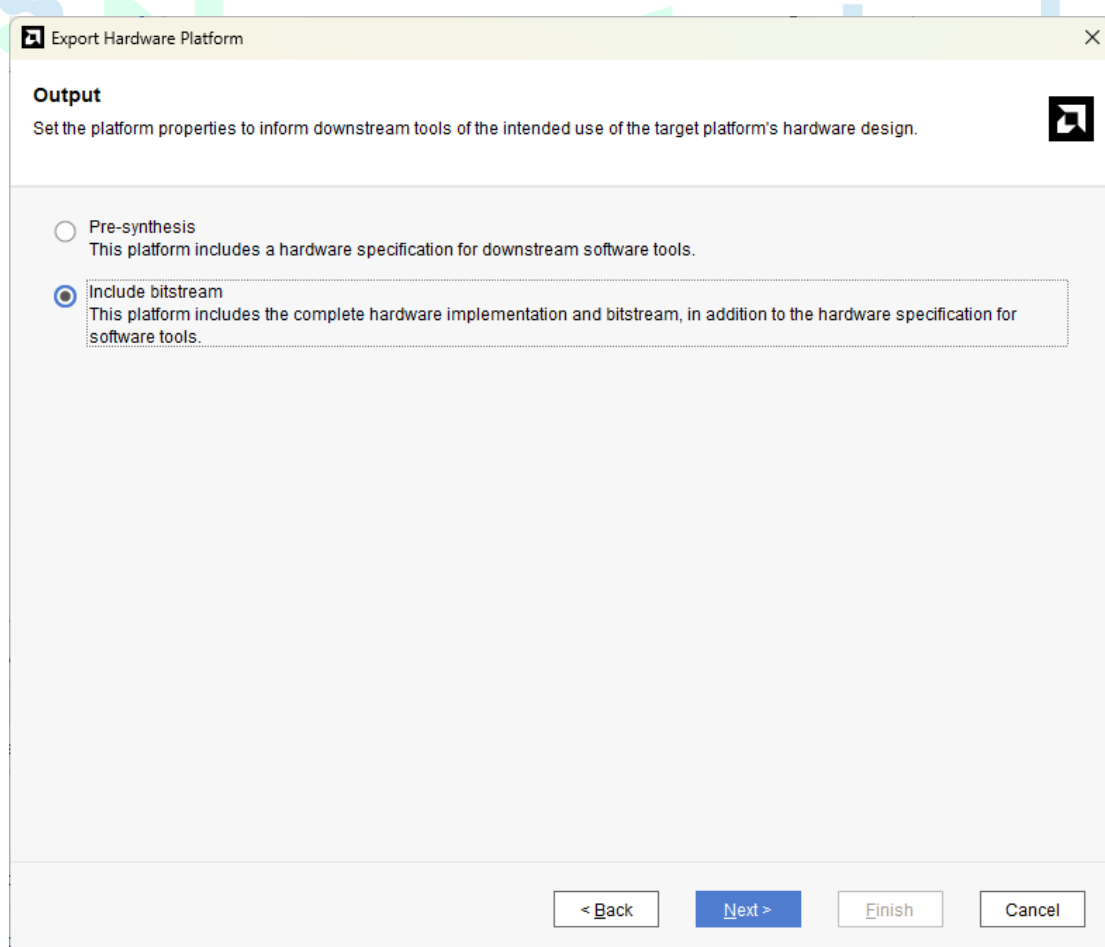
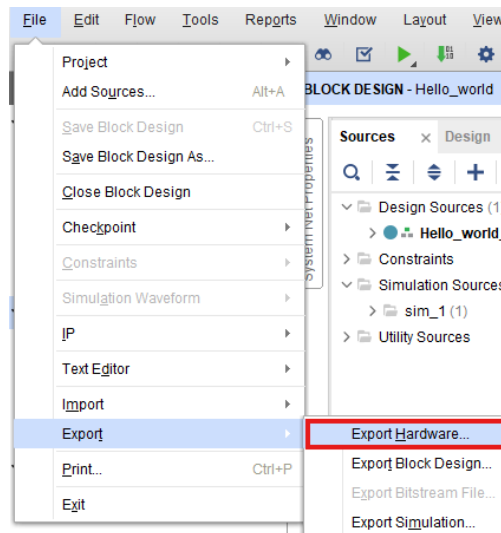
Step 10:

Click “Generate Bitstream” under the “Program and Debug” section to synthesize, implement, and generate a bitstream.



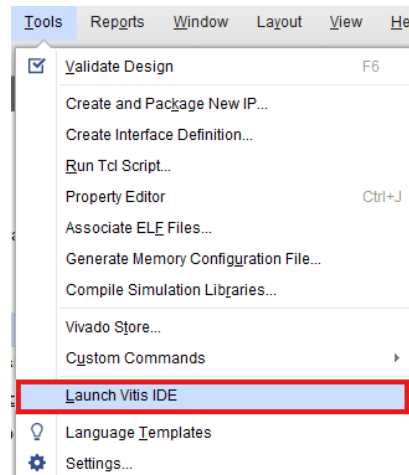
Step 11:

Once the implementation and generation of the bitstream are completed, we need to export the hardware along with the bitstream. Go to the “File” menu and select “Export->Export Hardware “. Select the “Include bitstream” checkbox and click “OK” in the “Export Hardware” wizard.

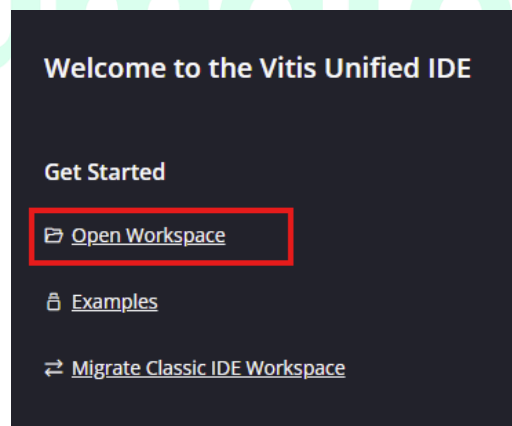


Step 12:

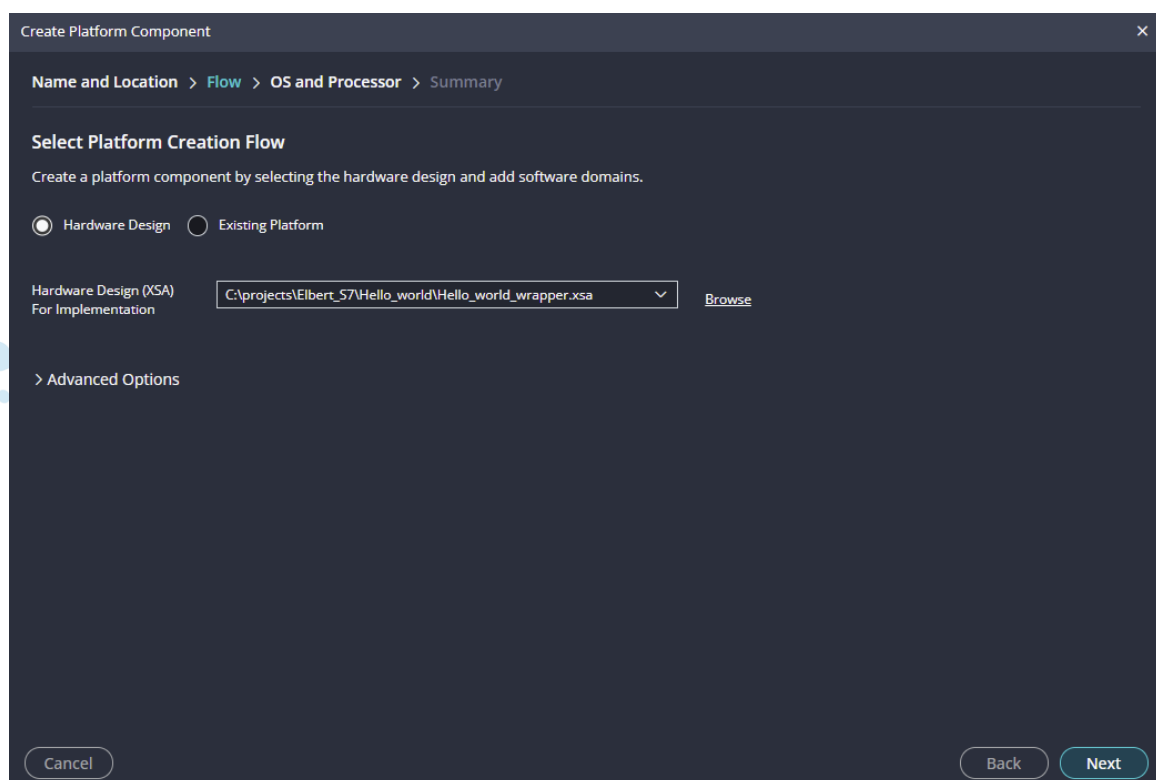
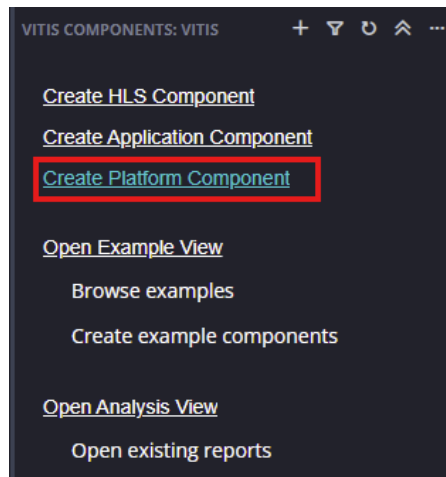
Select Launch Vitis IDE from the Tools menu.

**Step 13:**

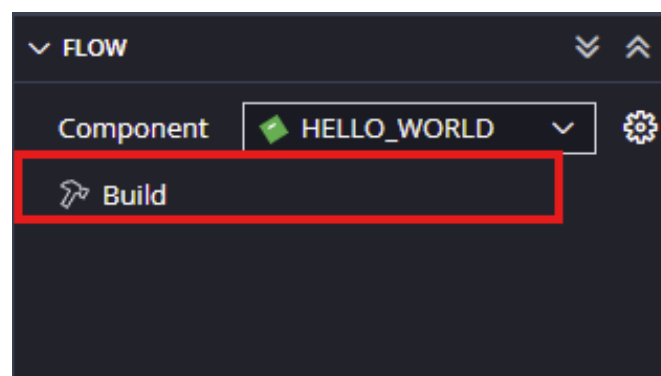
After Vitis Unified IDE window opens, click on “Open Workspace” and select necessary folder to keep the Vitis files.

**Step 14:**

Create a new platform for the project, by selecting “Create Platform Component”, click “Next”, in the Flow tab select the XSA file saved using the step 11 and finally click “Next” and “Finish” respectively.

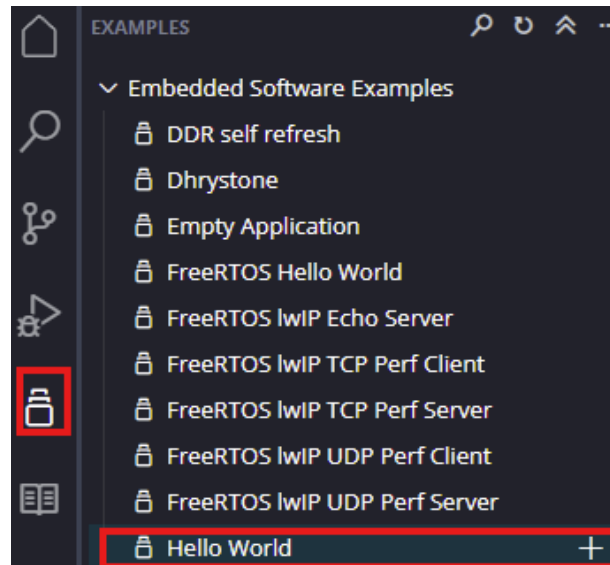


After successful creation of the platform, build the platform.

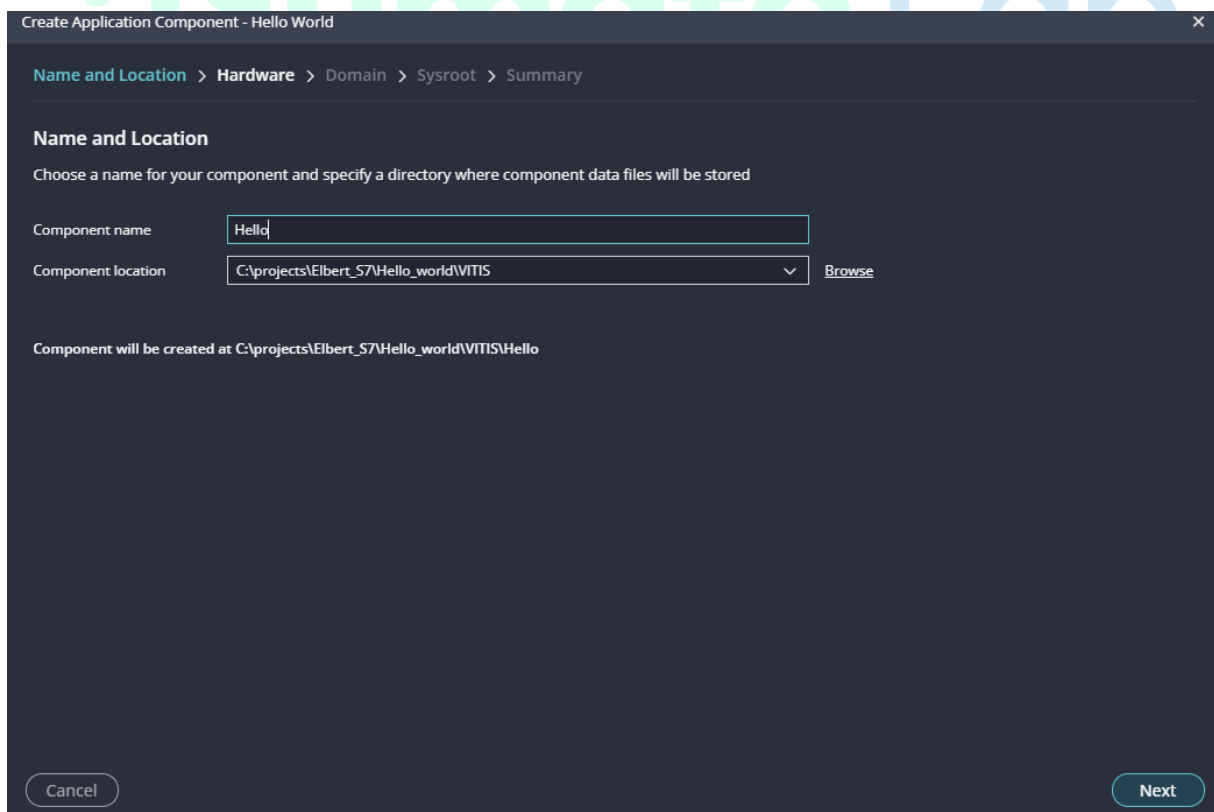


Step 15:

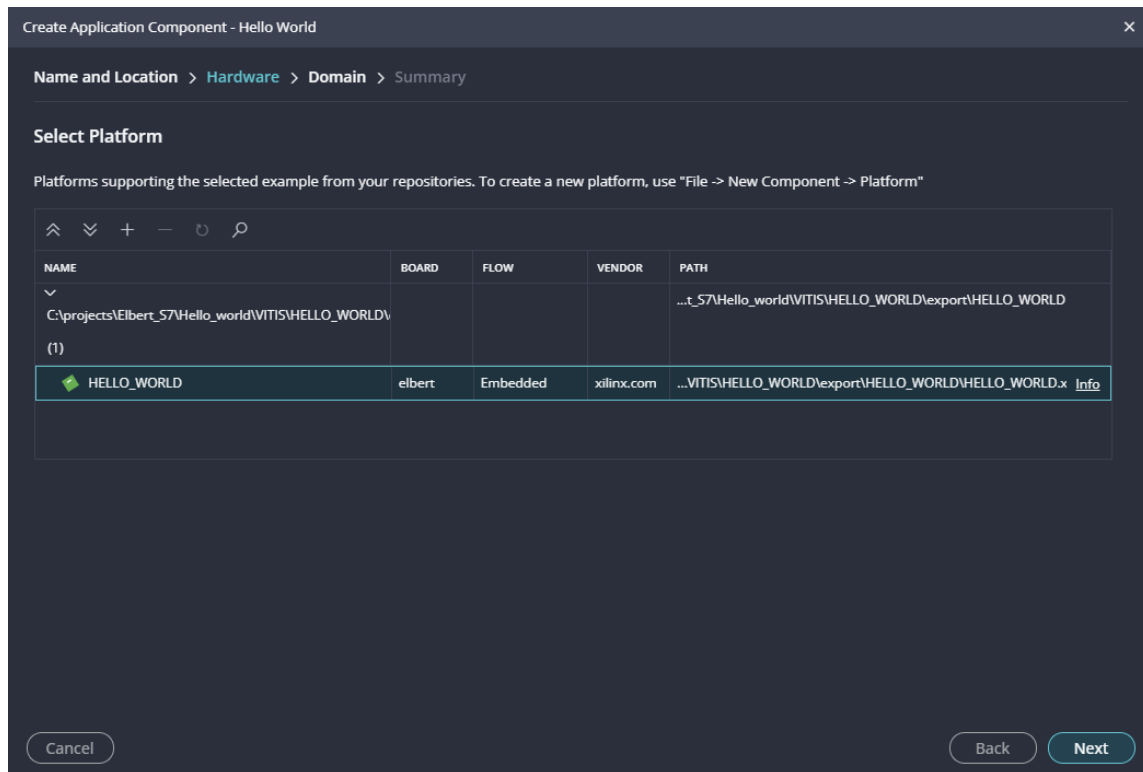
Next create the Hello world Application component by selecting the “Hello world” template from the “examples”,



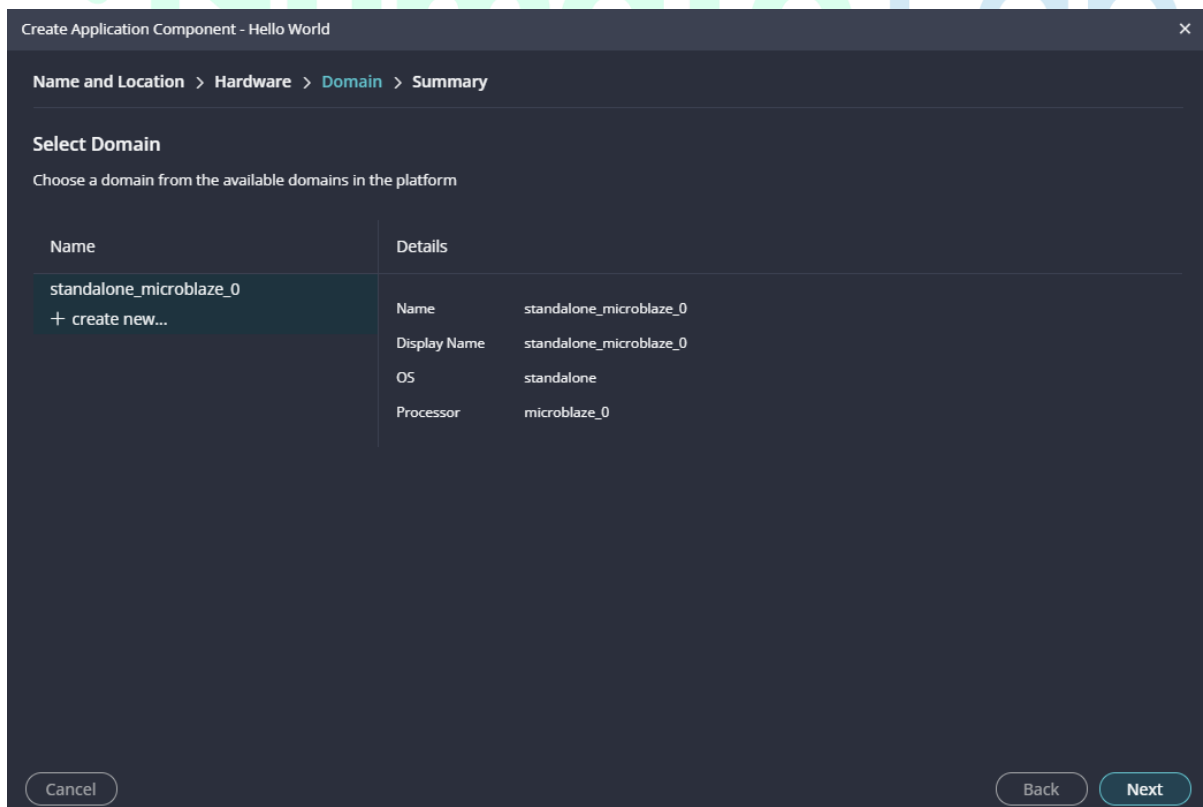
In “Create Application Component” tab specify project name and location, click “Next”.



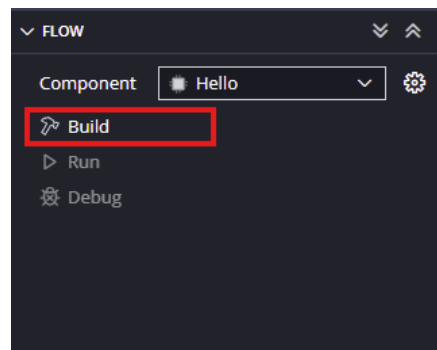
Select newly created Platform and click “Next”.



Select the domain as “Standalone_microblaze_o” and click “Next” and click on “Finish”.

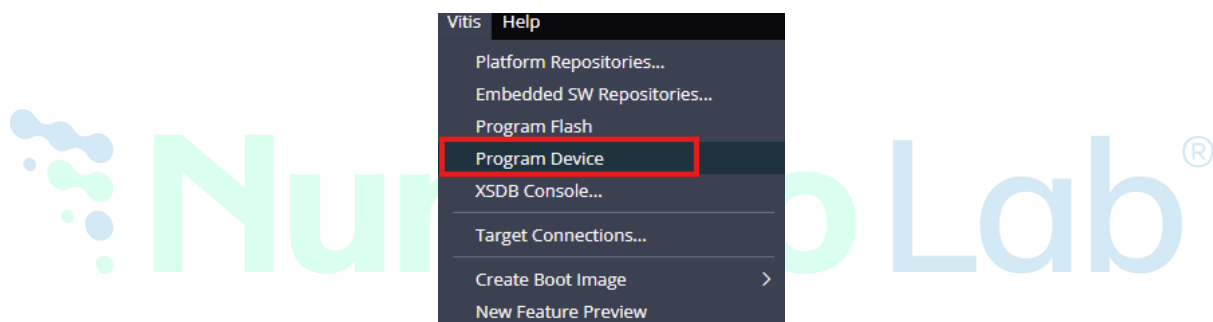


When the Helloworld project is added successfully, build the project manually.

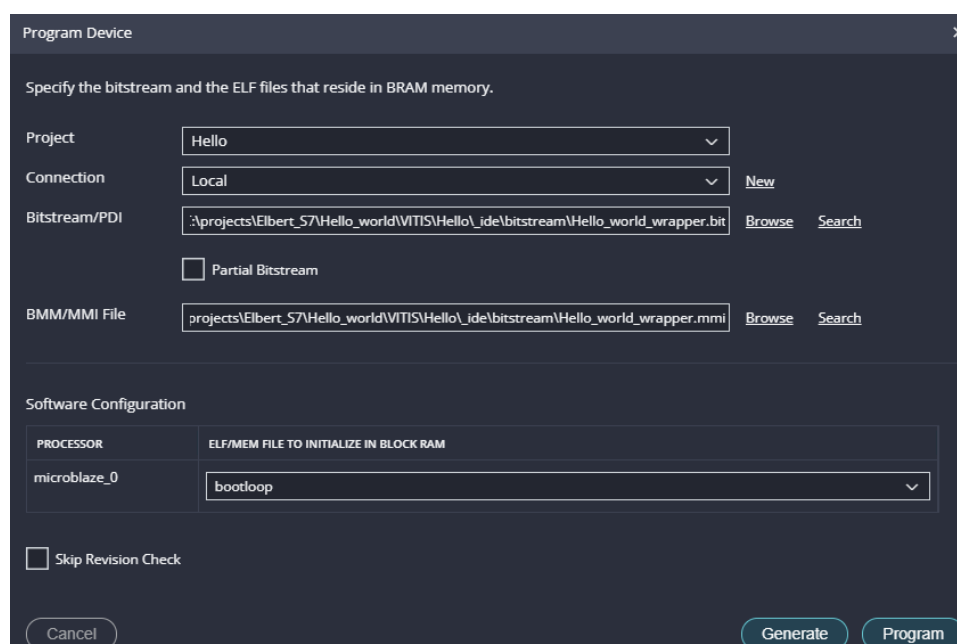


Step 16: Once the build is completed successfully, power up Elbert S7 FPGA Development Board using USB type C cable.

Step 17: Program the FPGA on Elbert S7 with a simple boot loop program by selecting the Program Device option from the Vitis menu.

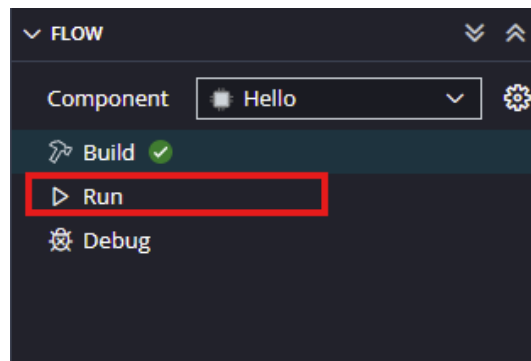


Once the “Program Device” window opens click on “Program “.

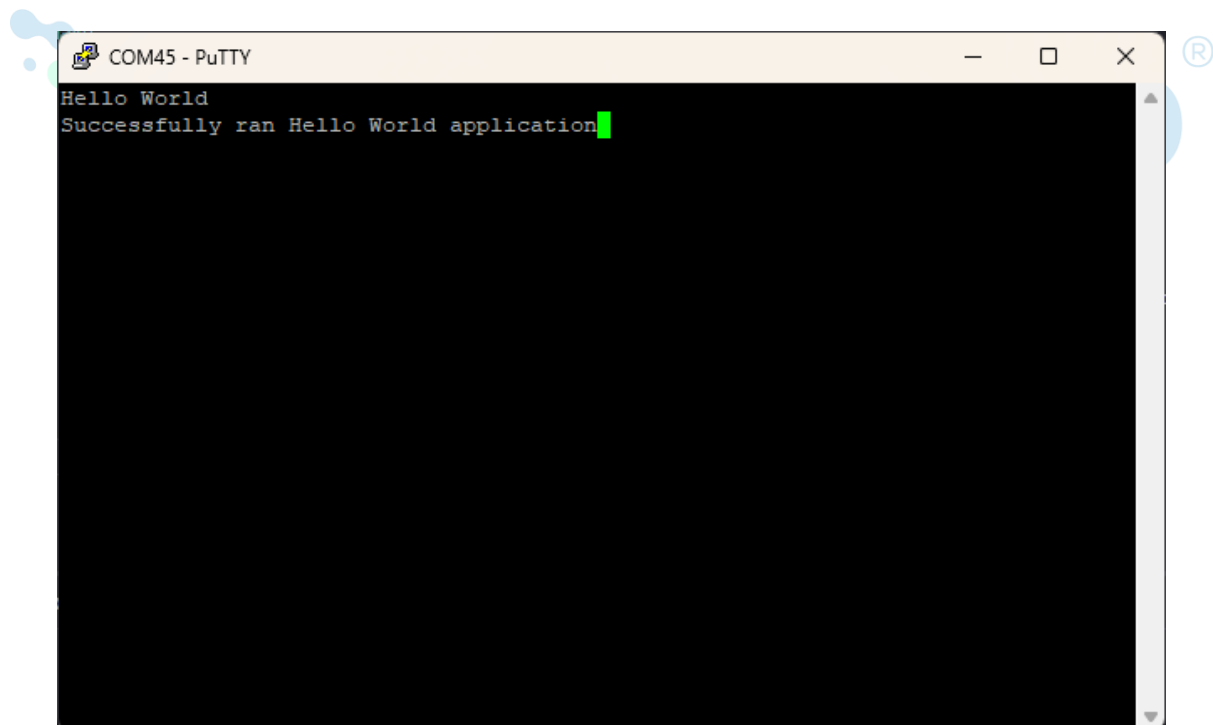


Step 18:

Meanwhile, open any serial terminal program (such as PuTTY, Teraterm etc) and open the port corresponding to Elbert S7 with a 9600 baud rate (the default baud rate given in UART IP). Program the board by selecting the “Run.”

**Step 19:**

If everything went well, the application running on the board should print “Hello World” over the UART and should be displayed on the Serial Terminal application.



2. Controlling Onboard Peripherals (LEDs, Seven Segment Display, and PMOD) with Slide Switches

INTRODUCTION

This project demonstrates how to interface and control key onboard peripherals- **LEDs, Seven Segment Display, and PMOD connectors**-using the slide switches on the Elbert S7 FPGA board. It serves as a foundational experiment to understand how input signals (from switches) can control output peripherals in an FPGA-based system.

The design includes predefined behaviours:

- When **no switch is active**, the **seven-segment displays** continuously count from 0 to 9 (same digit on all displays), while the **LEDs** perform a running light pattern.
- When **specific switches are turned ON**, the system displays corresponding values:
 - **Switch 1 (SW0)**: The seven-segment shows **3**, and its binary representation (**00000011**) is shown on the LEDs.
 - **Switch 2 (SW1)**: The seven-segment shows **5**, and the LEDs display **00000101**.
 - **Switch 3 (SW2)**: The seven-segment shows **7**, and the LEDs display **00000111**.
- When **Switch 4 (SW3)** is turned ON, **all PMOD pins go high**, enabling external module activation or signalling.

This project is designed as a basic interface demo. You are encouraged to **modify the functionality**-such as changing the numbers shown on the seven-segment display, customizing LED patterns, or controlling PMOD outputs differently-to suit your specific application needs or to experiment with your own designs.

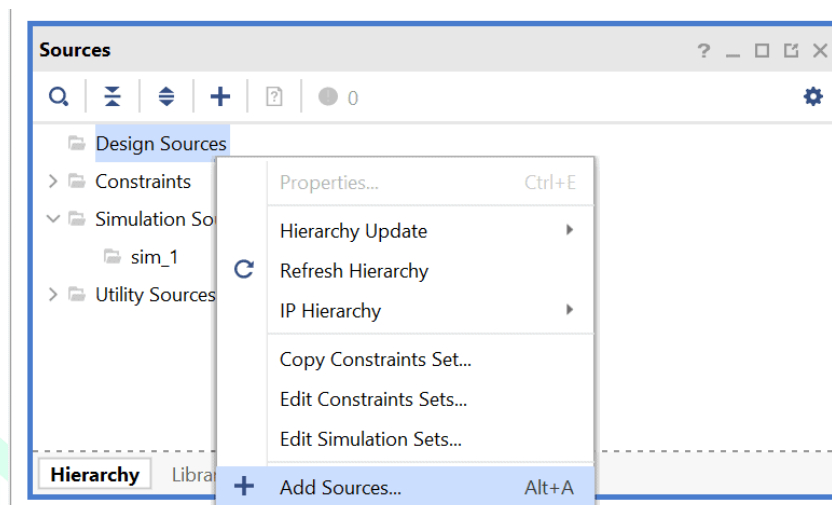
By completing this project, you'll gain hands-on experience with I/O interfacing, logic control based on inputs.

STEP 1:

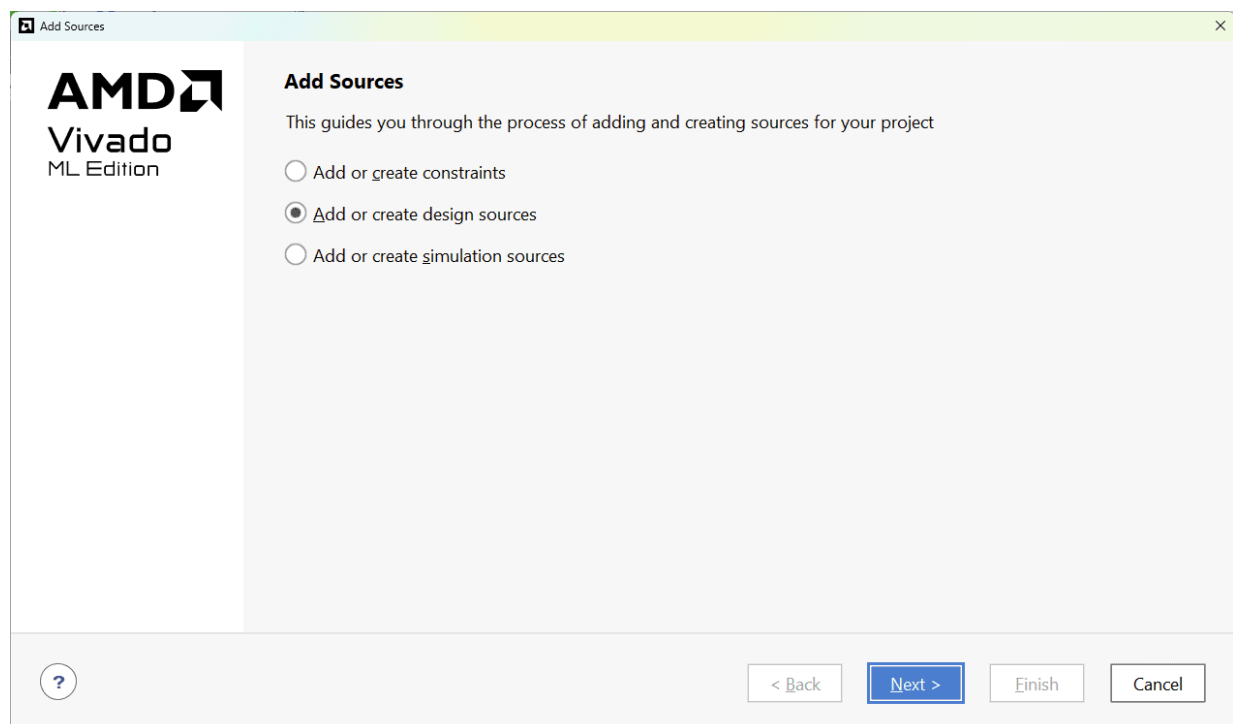
To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section “[Getting Started with Vivado: Creating a New FPGA Project.](#)”

STEP 2:

After creating a new project successfully, In the Sources tab, Right-click ‘**Design Sources**’ and click ‘**Add Sources**’. It will open a new ‘**Add Sources**’ window.

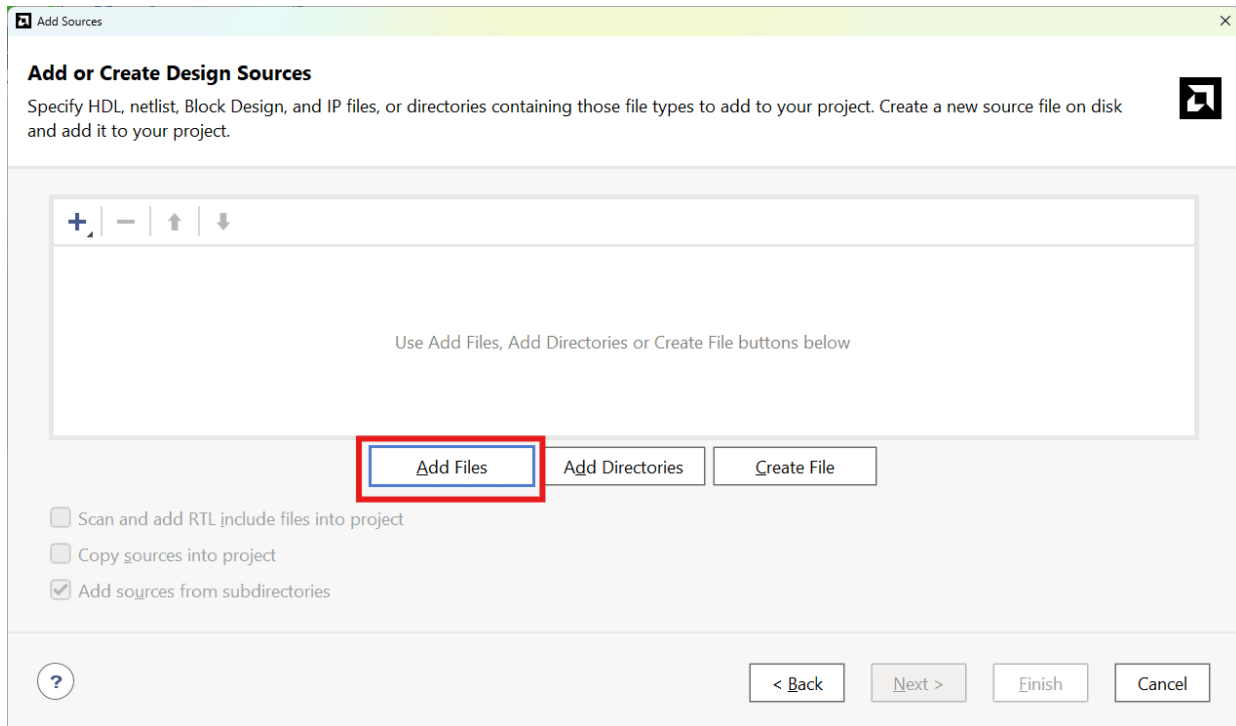


Once the “**Add Sources**” window opens select “**Add or create design sources**” and click on “**Next**”

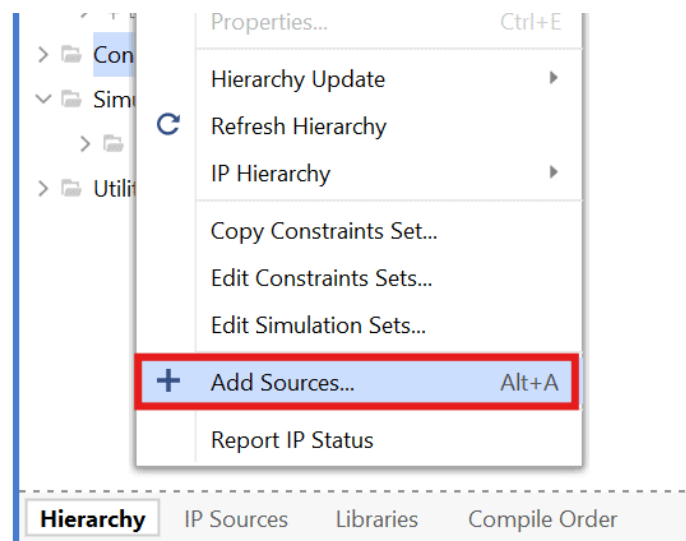


Step 3:

Download and extract the RTL source files from [here](#) and add them to the project by selecting ‘**Add Files**’ in the “**Add or Create Design Sources**” tab and click on “**Finish**”.

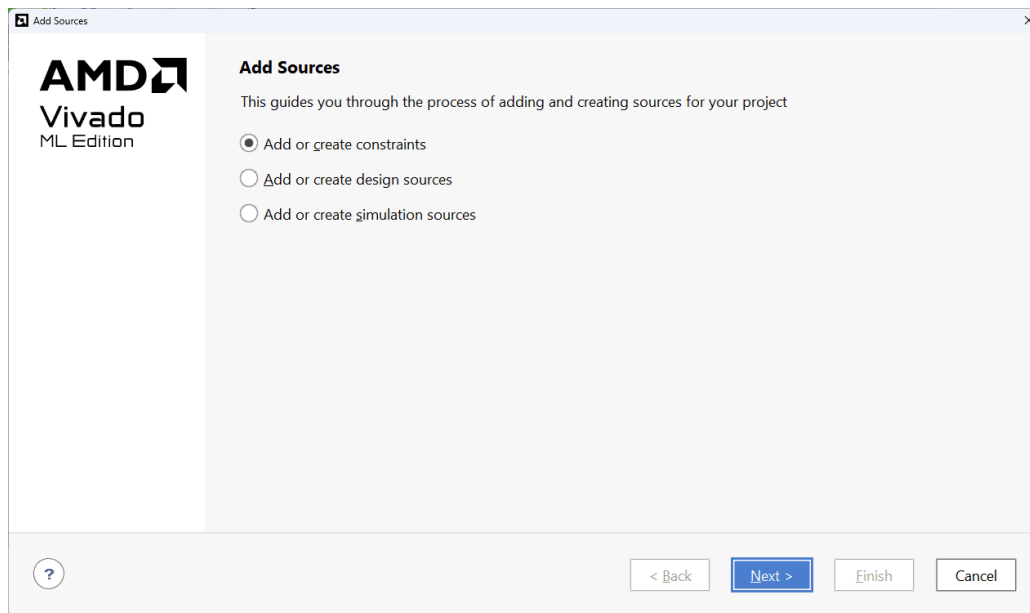
**Step 4:**

In Sources tab of Vivado, Right-Click on ‘**Constraints**’ and click ‘**Add Sources**’.

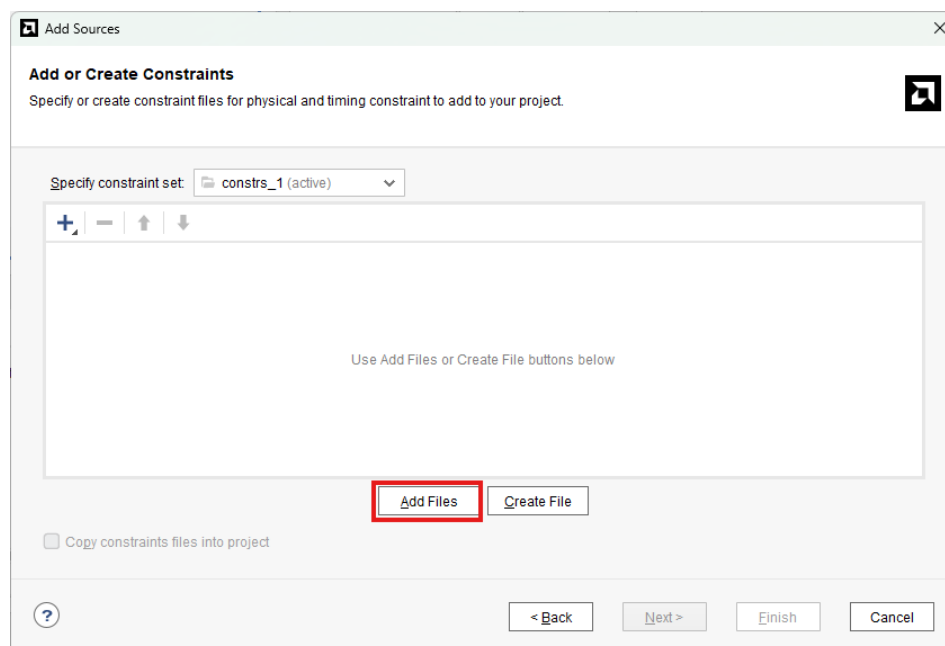


Step 5:

Once the “**Add Sources**” tab opens select “**Add or create constraints**” and click on “**Next**”.

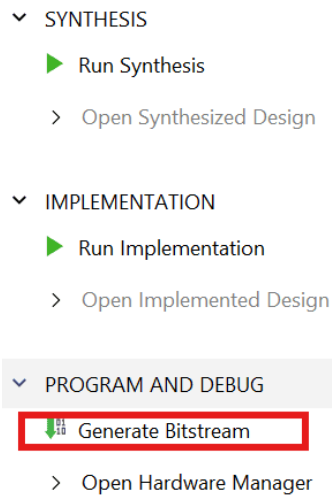
**Step 6:**

Download and extract the xdc file from [here](#) and add them to the project by selecting ‘**Add Files**’ in the “**Add or Create Constraints**” tab and click on “**Finish**”.

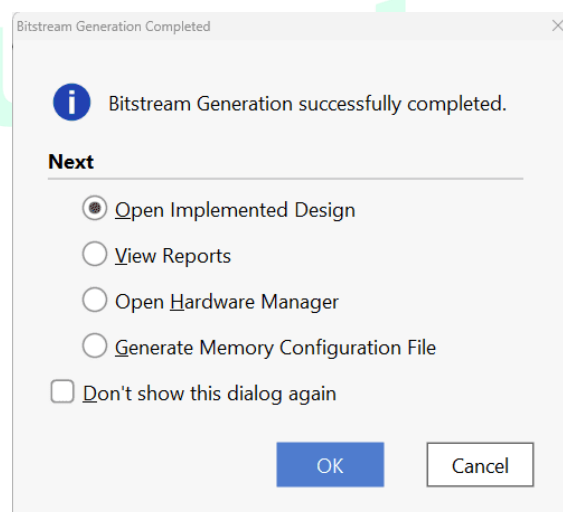


Step 7:

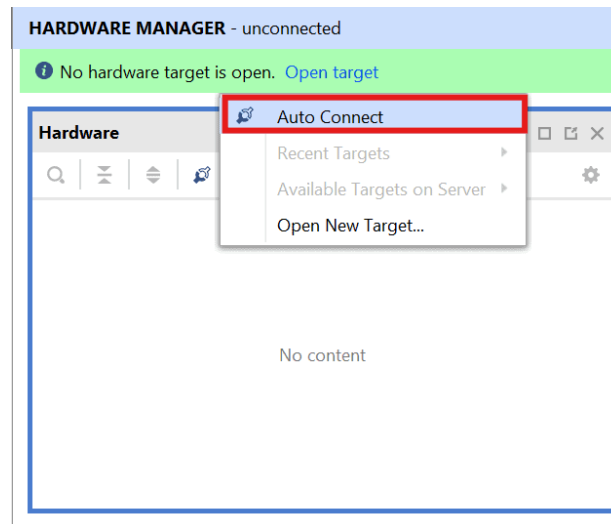
In Project Manager tab, click on '**Generate Bitstream**'.

**Step 8:**

Once the bitstream is successfully generated, close any “**Bitstream Generation Completed**” dialog which comes up asking for what to do next.

**Step 9:**

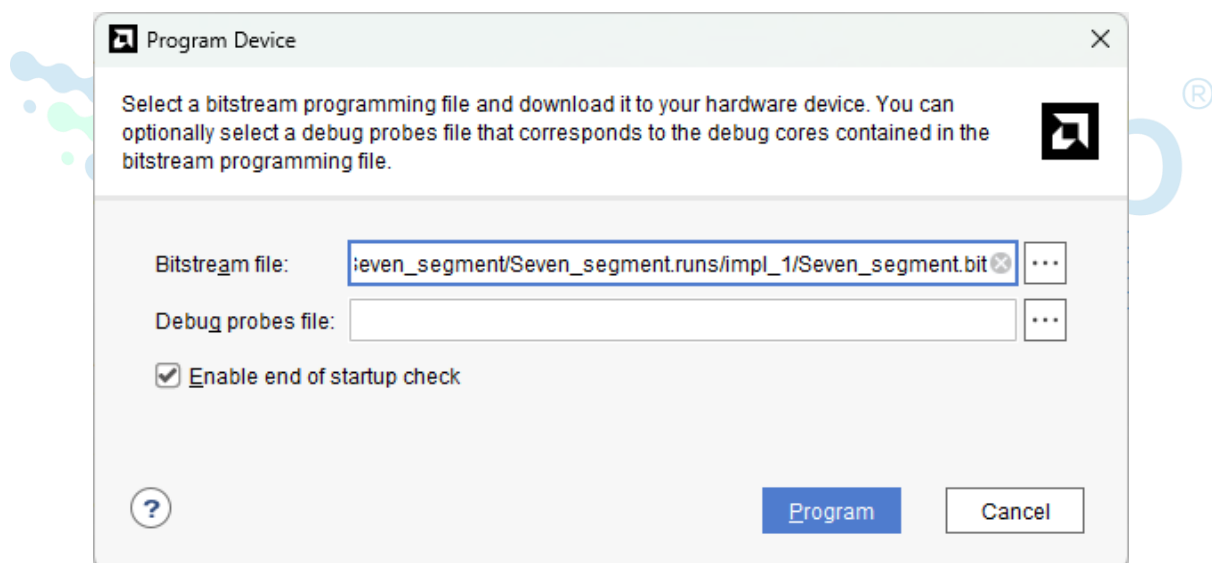
Click on '**Open target**' and '**Auto Connect**'.

**Step 10:**

Right Click on the device (**xc7s50_o**) and select “**Program Device**” option.

Step 11:

Click “**Program**” and observe the output.



3. HDMI OUTPUT EXAMPLE DESIGN

INTRODUCTION

HDMI (High-Definition Multimedia Interface) represents a significant advancement over the older VGA standard by offering a digital solution that integrates both high-resolution video and audio into a single interface. Unlike VGA, which relies on analog signals, HDMI uses digital transmission to deliver superior picture and sound quality. HDMI achieves this by transmitting pixel data serially at ten times the pixel clock frequency through TMDS (Transition Minimized Differential Signalling). This method reduces the number of signal transitions, which helps minimize potential data errors and maintains signal integrity.

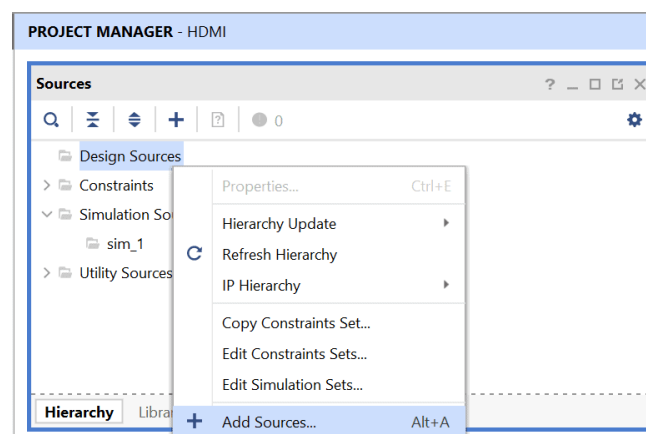
The tutorial focuses on demonstrating DVI-D output through the Elbert S7 FPGA Module. DVI-D, or Digital Visual Interface – Digital, is a variant of HDMI and shares the same electrical and physical layer specifications. As a result, HDMI cables can also carry DVI-D signals, meaning that HDMI monitors are fully compatible with DVI-D outputs.

STEP 1:

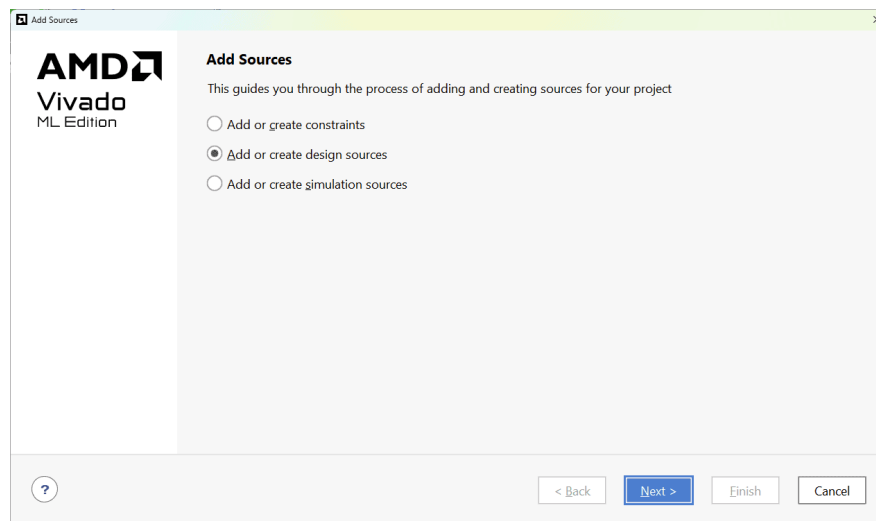
To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section “[Getting Started with Vivado: Creating a New FPGA Project.](#)”

Step 2:

After creating a new project successfully, then in the Sources tab, Right-click ‘**Design Sources**’ and click ‘**Add Sources**’. It will open a new ‘**Add Sources**’ window.

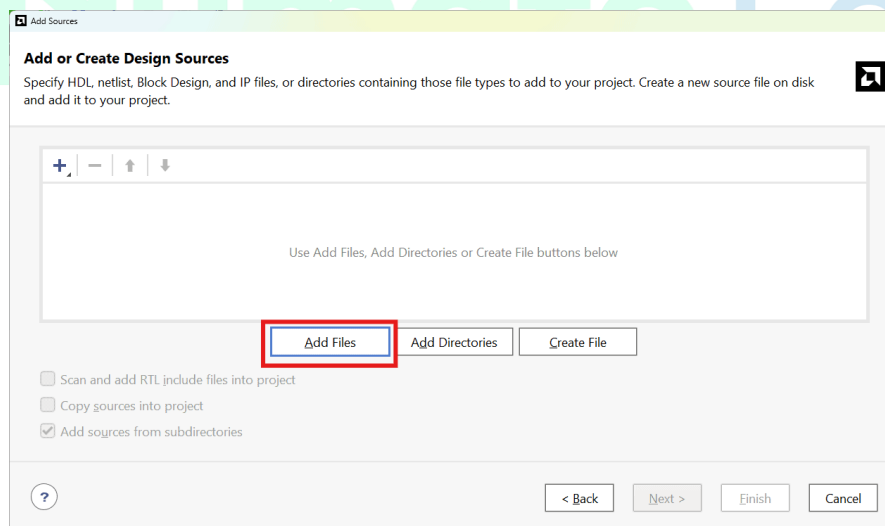


Once the “**Add Sources**” window opens select “**Add or create design sources**” and click on “**Next**”



Step 3:

Download and extract the RTL source files from [here](#) and add them to the project by selecting ‘**Add Files**’ in the “**Add or Create Design Sources**” tab and click on “**Finish**”.



The HDMI interface has 3 pairs of differential data signals and 1 pair of differential clock signals:

- `data_p[2:0]` & `data_n[2:0]` : These are HDMI/DVI differential signals carrying the video data to be displayed on screen.
- `clk_p` & `clk_n` : HDMI pixel clock differential pair of signals.

First, VGA signals are generated inside vga module. Then the VGA signals are encoded to 10-bits per channel and the data is then serialised to 10x of pixel clock rate. Finally, the three channels along with pixel clock are driven out using TMDS differential drivers.

In the top module (dvid_test), the two submodules dvid and vga are instantiated. Clocking IP (clocking wizard) is used to generate clocks for VGA and DVI-D.

dvid_test: In this module, a “Clocking Wizard” IP core is instantiated to generate required clocks for VGA and DVI-D. A 100MHz clock from the onboard oscillator is provided as input, and following clocks are derived from it:

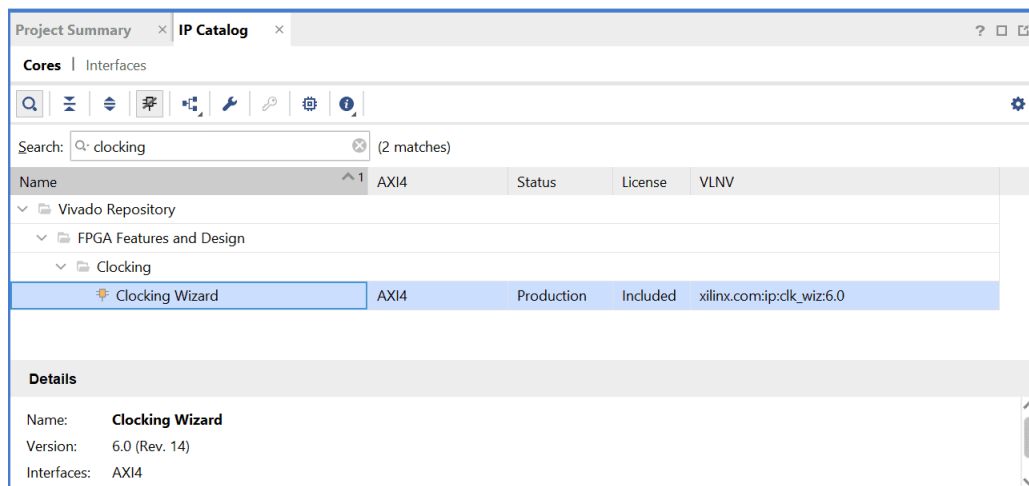
- clk_vga: 25MHz clock. This is the pixel clock frequency for 640×480@60Hz VGA resolution.
- clk_dvi & clk_dvin : 125 MHz clocks. clk_dvin is 180 degrees out of phase to clk_dvi. These clocks are used for serialization using ODDR2.

vga: VGA signals are generated in this module. This design generates VGA at 640×480@60 Hz resolution.

dvid: VGA signals and clocks are given as input to this module and the DVI TMDS signals are generated as the output. It uses TMDS_encoder module to generate TMDS signals. TMDS uses 8b/10b encoding in which the 8-bit color data (red, green & blue) generated in VGA module is converted to 10 bits. Then this data is serialised using ODDR2 (Double Data Rate primitive). The 10-bit TMDS data is generated at 25 MHz. ODDR2 uses 5 times the frequency of pixel clock (i.e. 125MHz) to serialize the 10-bit encoded data. Note that ODDR2 serialises 2-bits in 1 clock cycle of 125MHz clock. This serialised data is converted into differential signals in top module (dvid_test) using OBUFDS drivers.

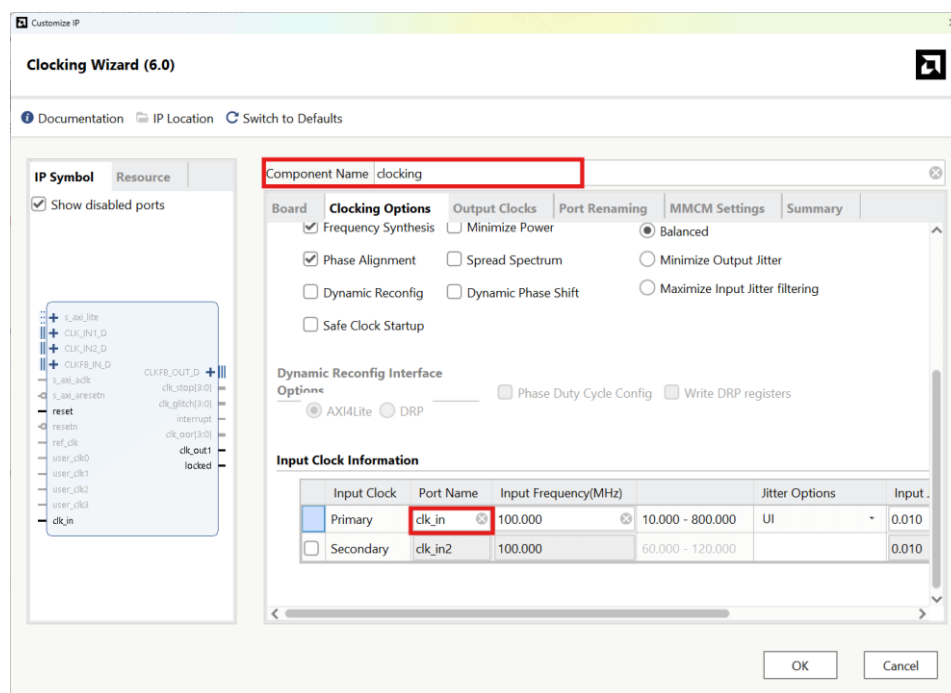
Step 4:

Add Clocking Wizard by clicking on IP catalog in Project Manager, type ‘**clocking**’ in search box and double-click ‘**Clocking Wizard**’ IP. It will open customisation window for ‘**Clocking Wizard**’.

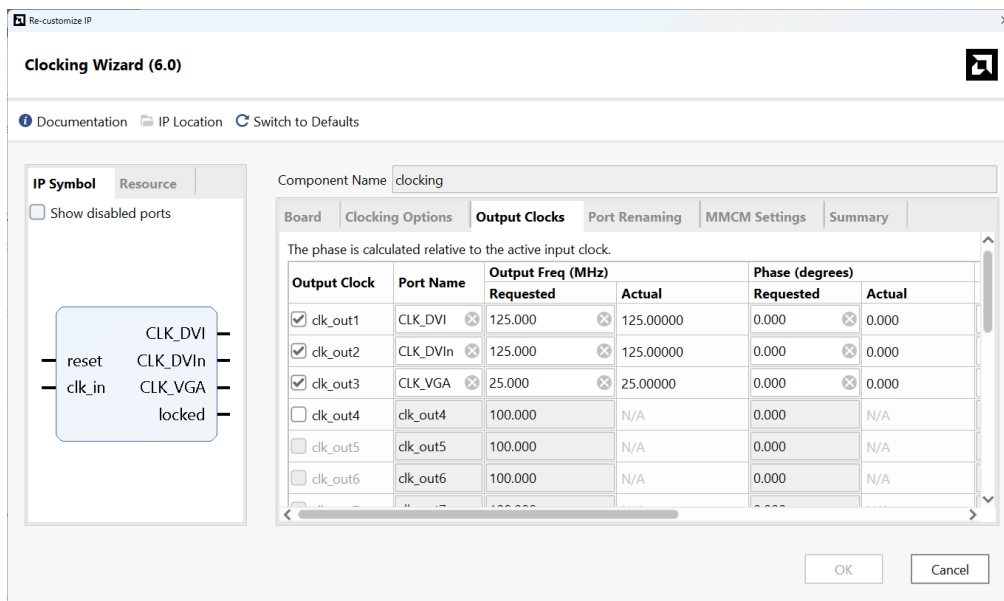


Step 5:

In 'Clocking Options' tab, give Component Name as 'clocking' and primary clock port name as 'clk_in'.

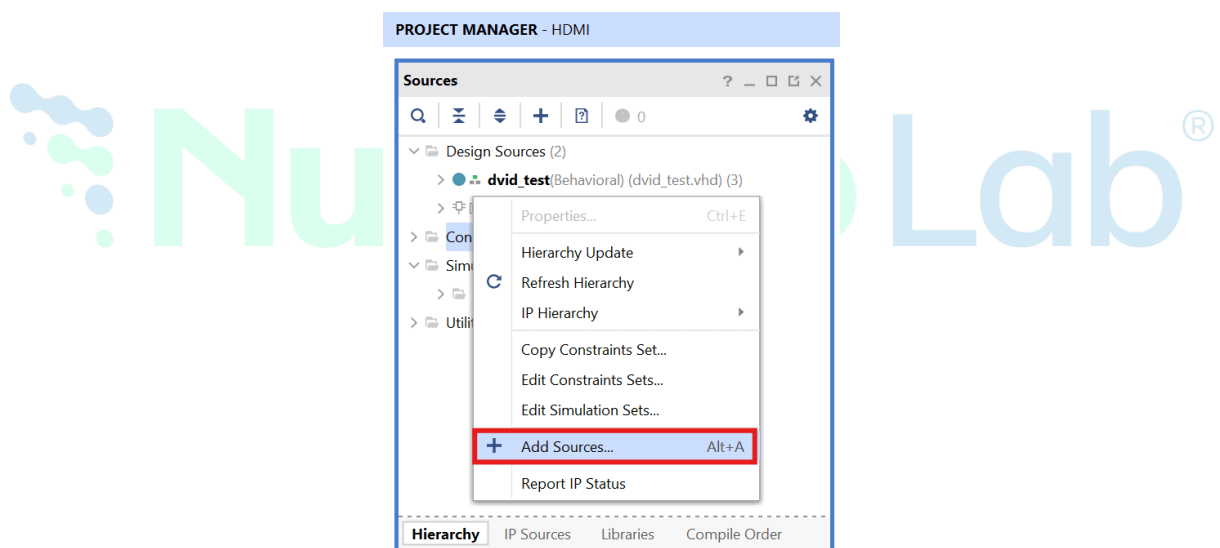


In 'Output Clocks' tab, enable 3 output clocks and provide their name, frequency as well as phase as shown in the image below. Click 'OK'.



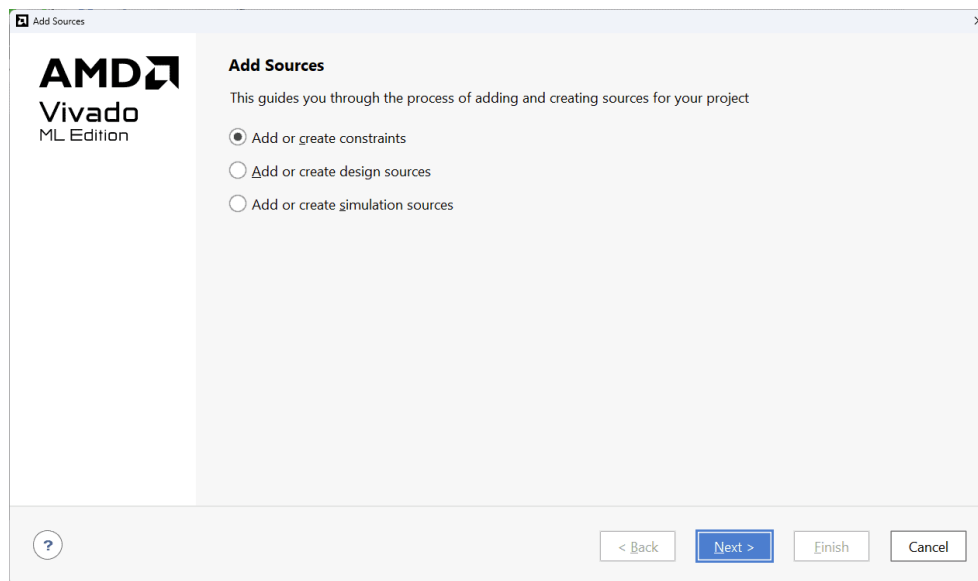
Step 6:

In Sources tab of Vivado, Right-Click on ‘**Constraints**’ and click ‘**Add Sources**’.



Step 7:

Once the “**Add Sources**” tab opens select “**Add or create constraints**” and click on “**Next**”.

**Step 8:**

Download and extract the XDC file from [here](#) and add them to the project by selecting ‘**Add Files**’ in the “**Add or Create Constraints**” tab and click on “**Finish**”.

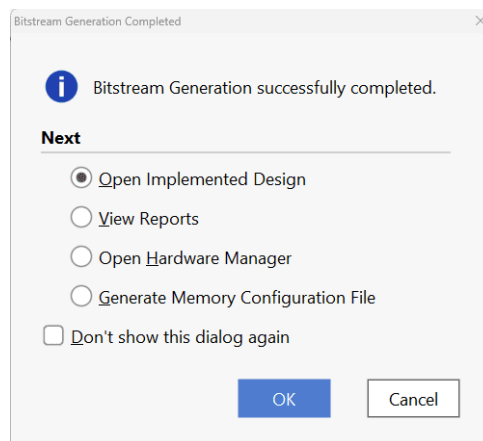
Step 9:

In Project Manager tab, click on ‘**Generate Bitstream**’.

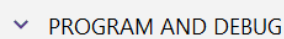
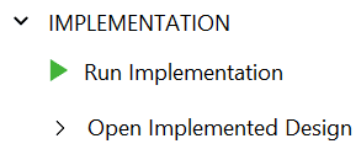
- ▼ SYNTHESIS
 - ▶ Run Synthesis
 - > Open Synthesized Design
- ▼ IMPLEMENTATION
 - ▶ Run Implementation
 - > Open Implemented Design
- ▼ PROGRAM AND DEBUG
 - ▶ **Generate Bitstream**
 - > Open Hardware Manager

Step 10:

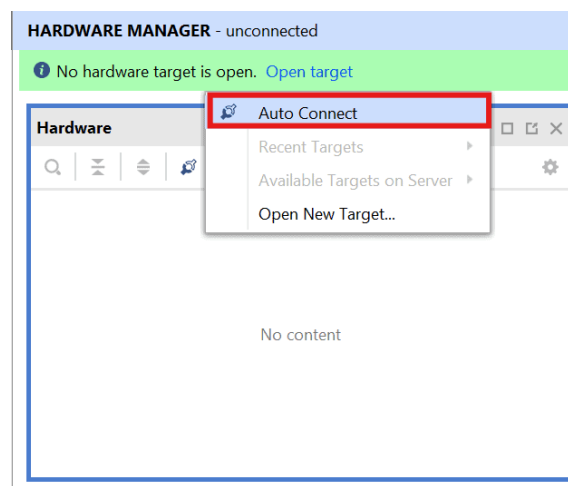
Once the bitstream is successfully generated, close any “**Bitstream Generation Completed**” dialog which comes up asking for what to do next.

**Step 11:**

Now click '**Open Hardware Manager**' to program the FPGA.

**Step 12:**

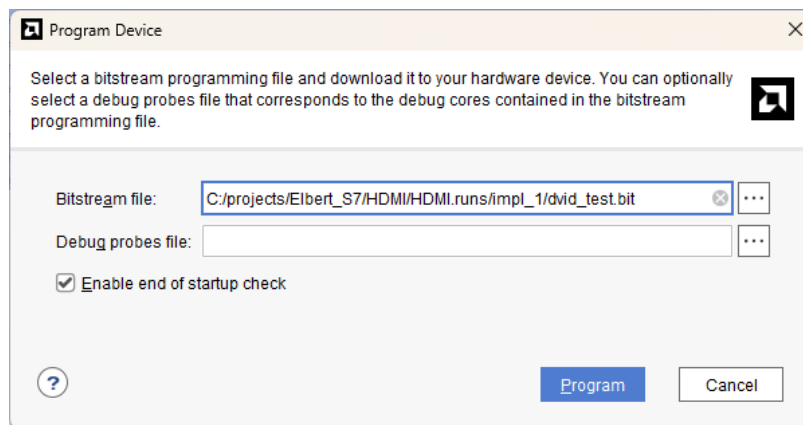
Click on '**Open target**' and '**Auto Connect**'.

**Step 13:**

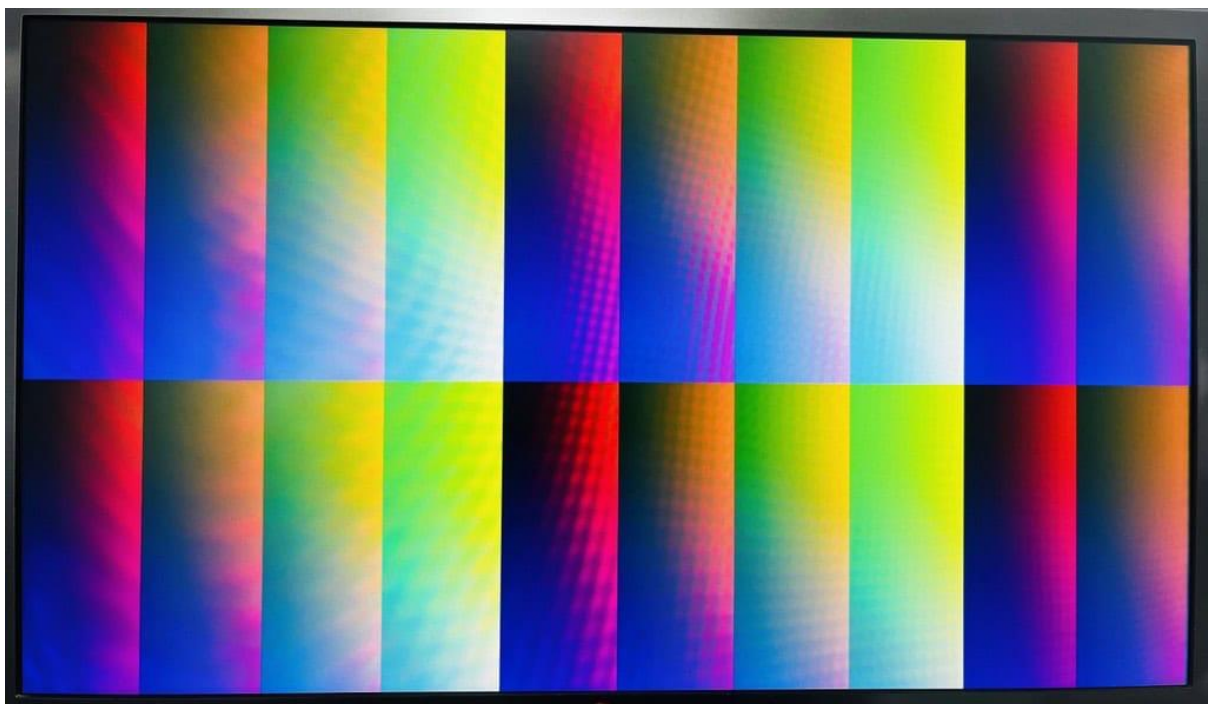
Right Click on the device (**xc7s50_o**) and select "**Program Device**" option.

Step 14:

Click “**Program**” and observe the output.

**Step 15:**

Once Elbert S7 is successfully programmed, it should begin generating HDMI signals and the monitor should display a colourful pattern at 640×480 @ 60Hz resolution.



That was it! You can play with the vga module to output different patterns and try to generate higher resolutions as well.

4. DDR3 Mem test on ELBERT S7

INTRODUCTION

This article aims to guide readers on how to effectively utilize the DDR3 memory available on the Spartan-7 FPGA using the AMD Memory Interface Generator (MIG) 7 IP Core. The MIG 7 IP core is a powerful tool provided by AMD that simplifies the process of interfacing with external DDR3 memory by handling the complex timing and calibration requirements.

When working with the MIG 7 IP core, users are presented with two interface options:

User Interface: This is a straightforward wrapper built on top of the Native Interface. It simplifies communication with the DDR3 memory by providing clear signal naming and intuitive data flow control.

AXI4 Interface: This interface follows the AXI4 protocol, which is widely used in FPGA designs for memory-mapped transactions. It allows seamless integration with AXI-based designs, providing better scalability and compatibility with IP cores that utilize the AXI protocol.

In this tutorial, we'll focus on how to test the DDR3 memory using the Memory Tests Template available in Vitis. This test template is a convenient way to verify memory functionality, ensuring that data can be reliably written to and read from the DDR3 memory. By following this guide, you'll gain insights into configuring the MIG 7 IP core, integrating it into your Vivado design, and performing practical memory tests on the Spartan-7 FPGA.

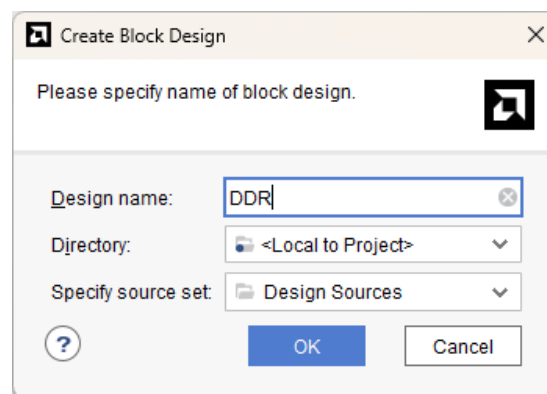
Whether you're a beginner or an experienced FPGA developer, this article will provide clear steps and explanations to help you successfully interface with DDR3 memory on the Spartan-7 platform.

STEP 1:

To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section “[Getting Started with Vivado: Creating a New FPGA Project.](#)”

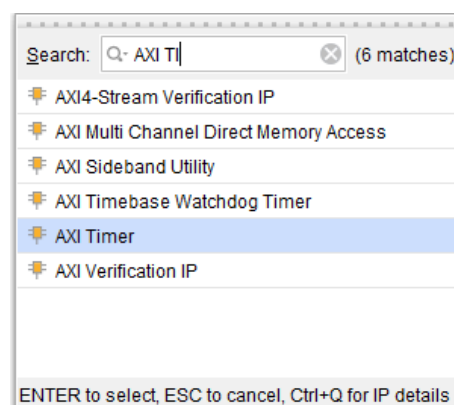
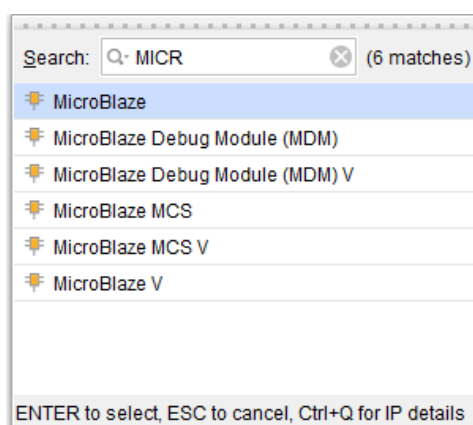
Step 2:

After creating a new project, Under the “Flow Navigator” panel, click “Create Block Design” under the IP Integrator section. Enter a name for the block design and click “OK”. An empty block design will be created.



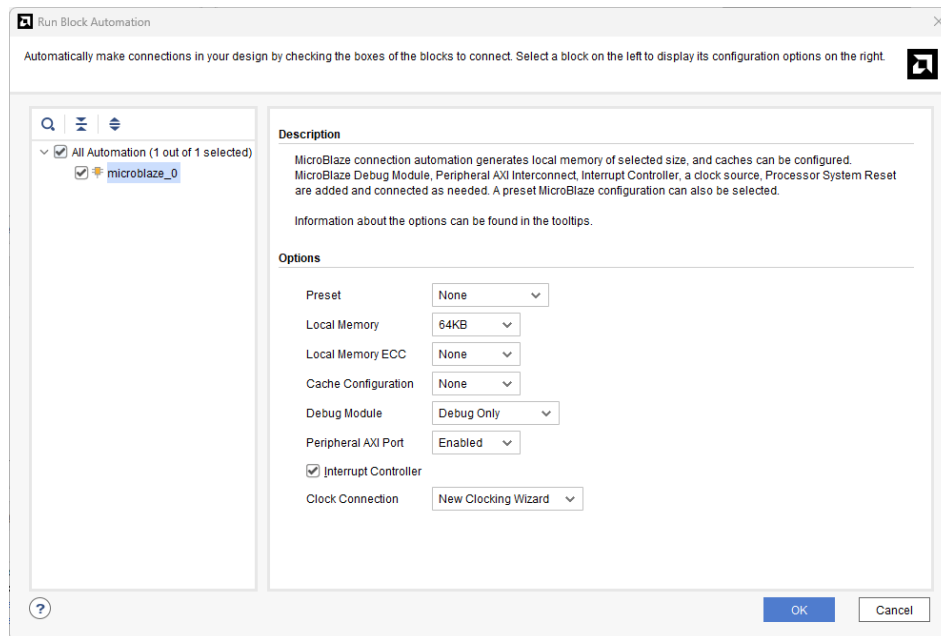
Step 3:

In the Diagram window, right-click and select “Add IP” from the popup menu. Search for “MicroBlaze” & “AXI Timer” and add them to the design by double-clicking them.

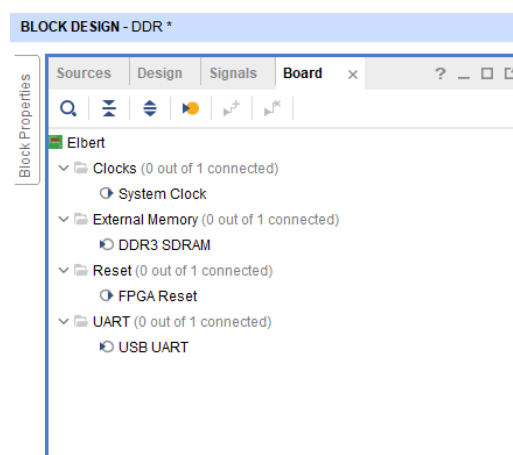


Step 4:

Click “Run Block Automation” present in the “Designer Assistance available” bar on the top left corner of the window to complete the design. Select the settings as shown in the following image.

**Step 5:**

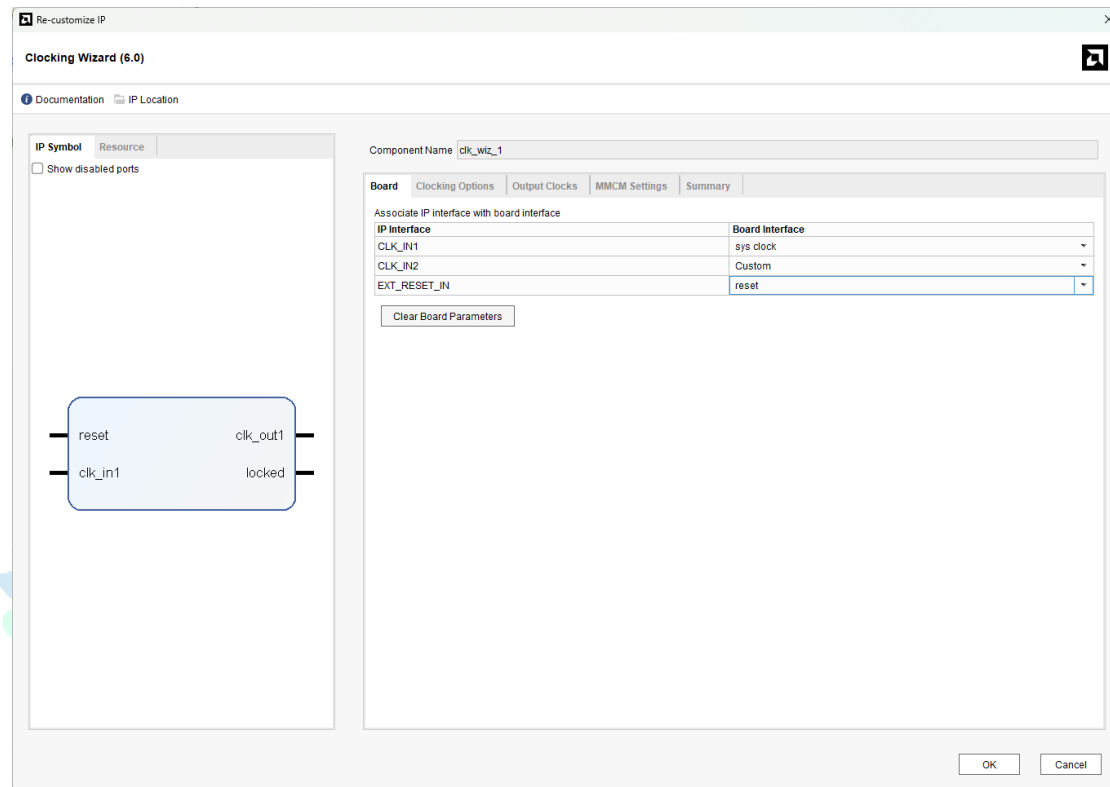
Click the “Board” tab. The default peripherals available for the Elbert board will be listed as shown below.

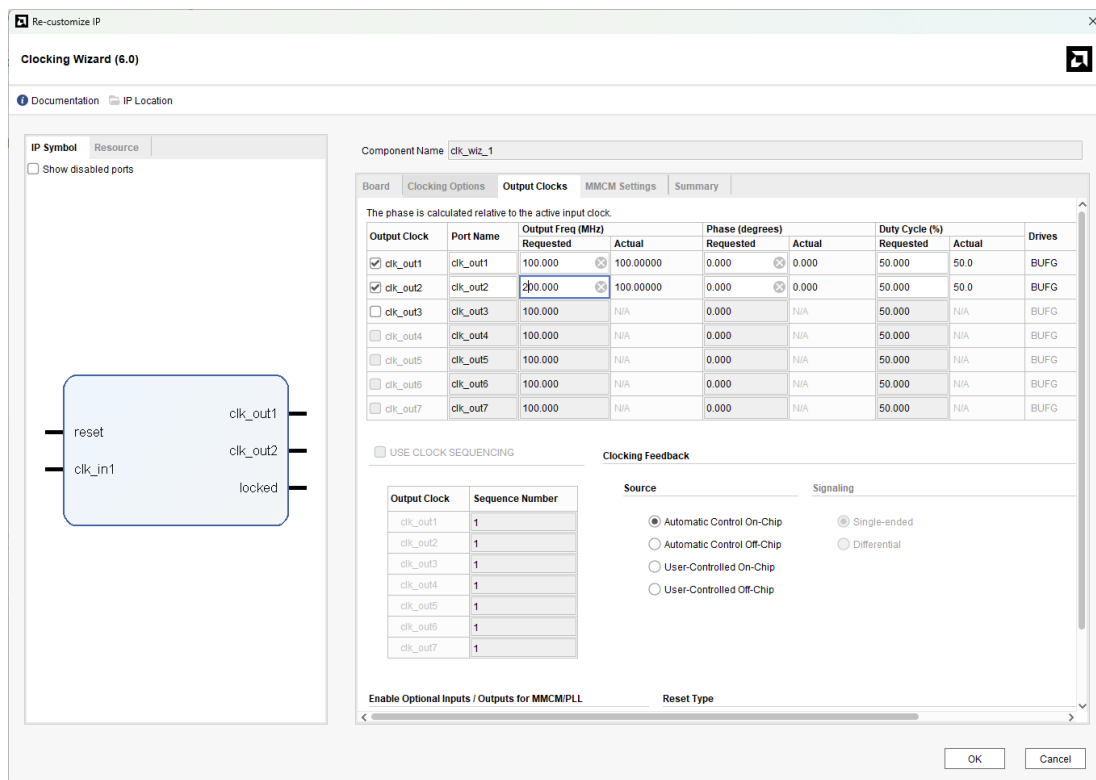


Add DDR3 SDRAM and USB UART to the design by double-clicking the corresponding peripherals.

Step 6:

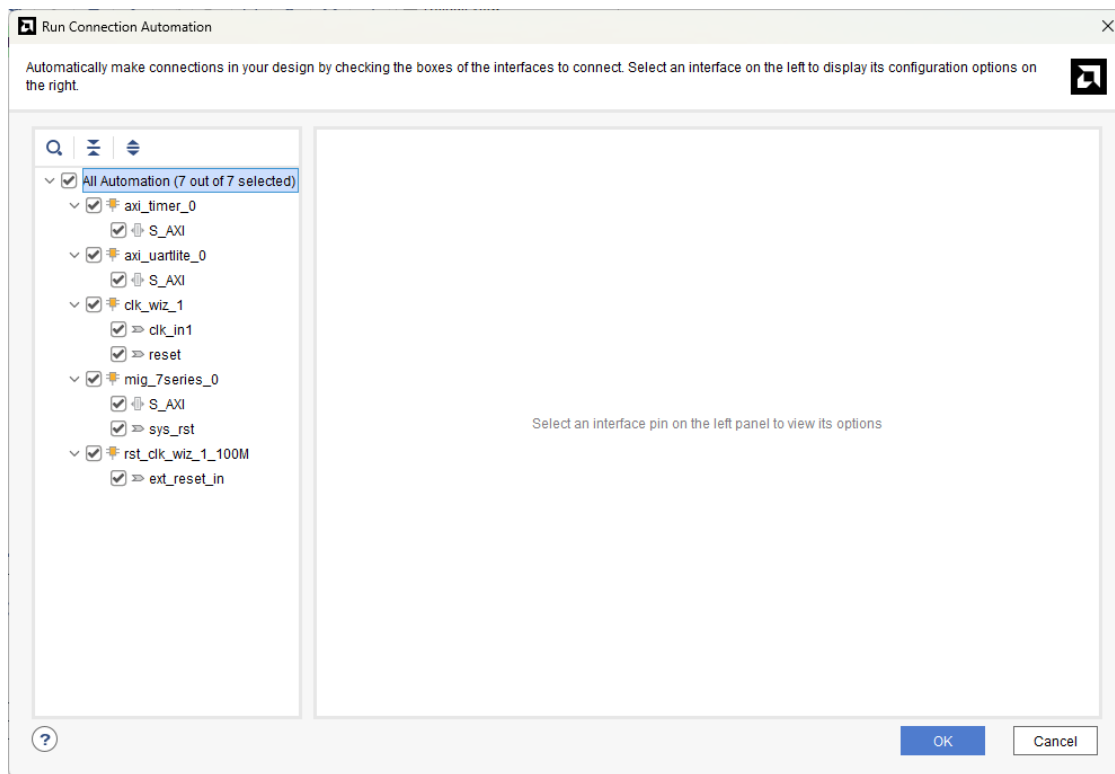
Double-click on the “Clocking Wizard” IP block and change the settings as shown below. In the “Output Clocks” section, set `clk_out1` frequency to 100 MHz and `clk_out2` to 200MHz.



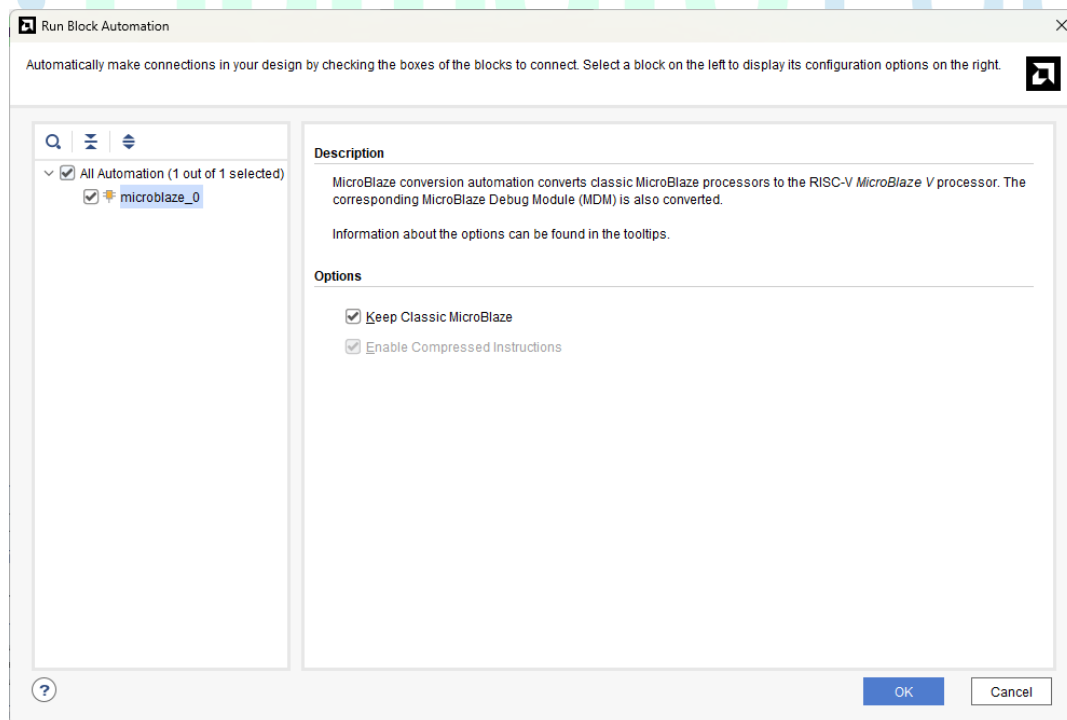


Step 7: Remove the existing connection to sys_clk_i of the “MIG 7 Series” block and connect it to clk_out2.

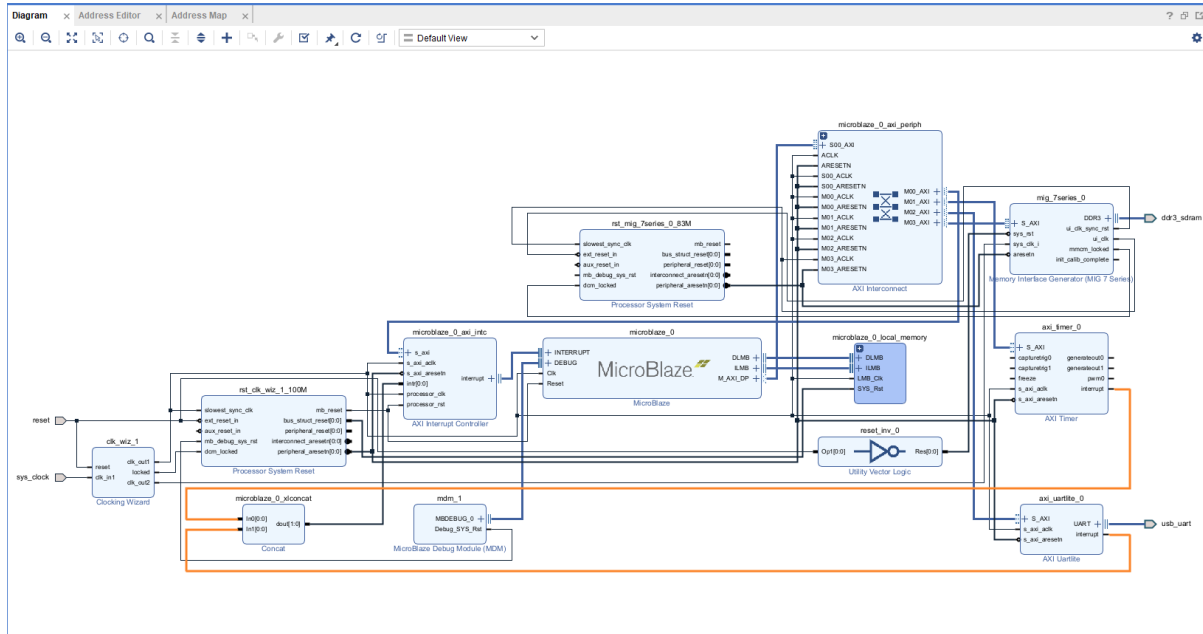
Step 8: Run “Connection Automation” so Vivado can connect the blocks to make a complete system.



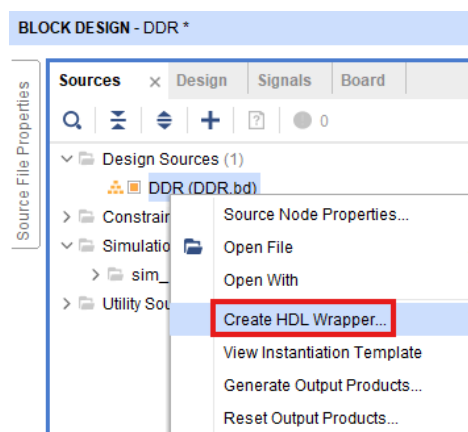
Click on "Run Block Automation" and select keep Classic Microblaze option as shown in the picture below.



Step 9: Connect interrupt output lines from “**AXI Timer**” and “**UARTLite**” to the “**Concat**” block as shown below figure. Select the “**Validate Design**” option from the **Tools** menu to make sure that connections are correct.

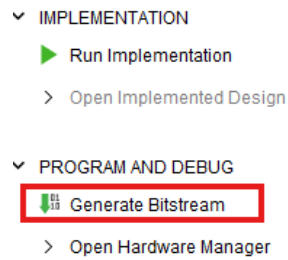


Step 10: Right-click “**ddr3**” in the “**Sources**” window and select “**Create HDL Wrapper**” from the popup menu. Click “**OK**” on the window that appears to finish generating a wrapper

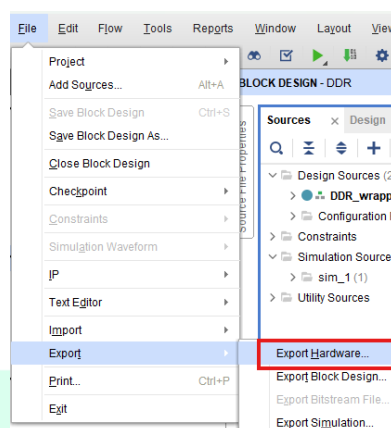


Step 11:

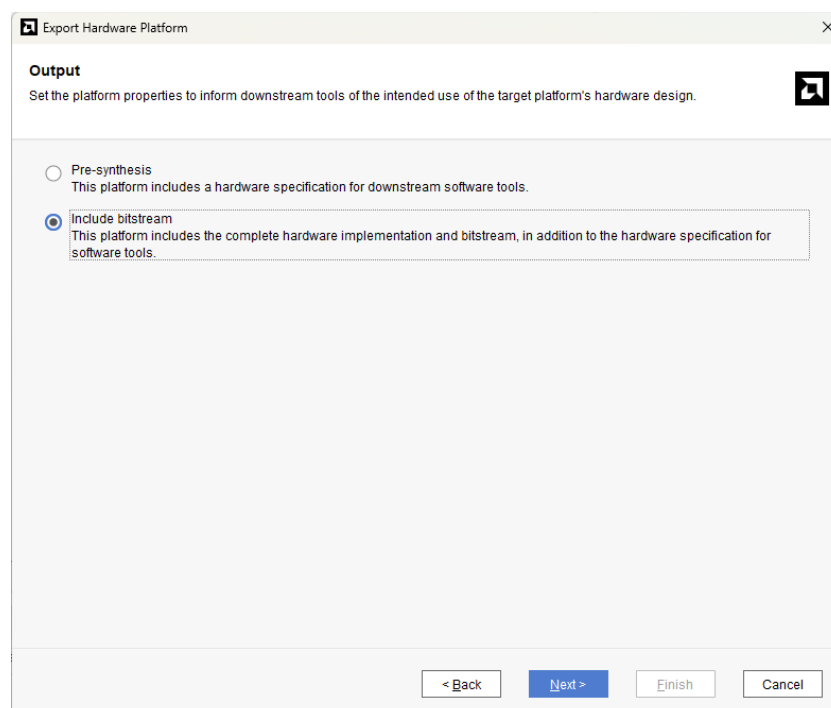
Click “**Generate Bitstream**” under the “**Program And Debug**” section to synthesize, implement and generate a bitstream.

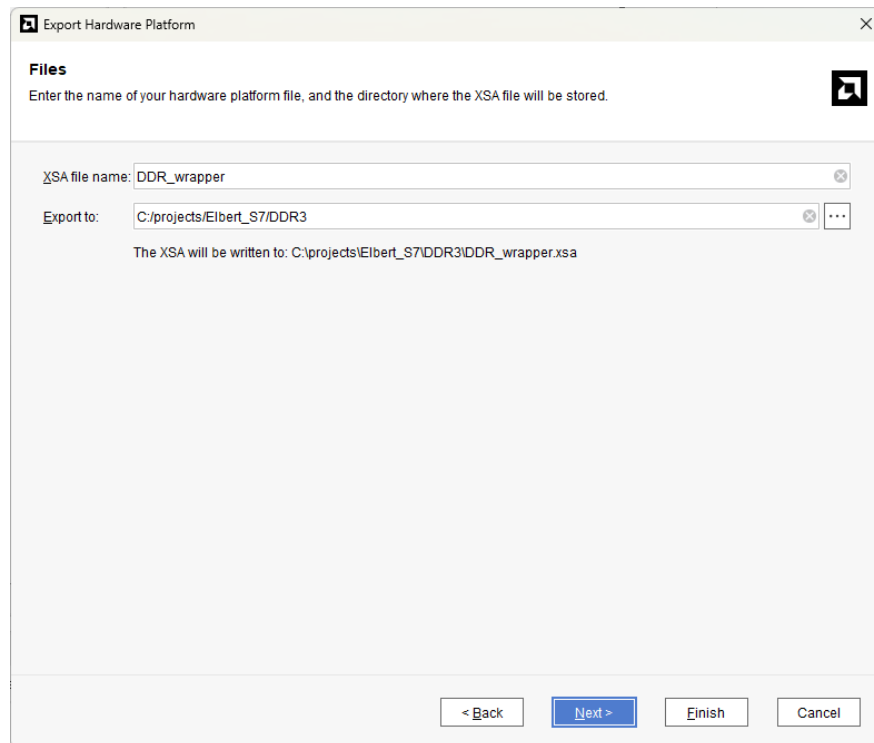


Step 12: After generating the bitstream successfully, select **Export -> Export Hardware** from the **File** menu. Click **Next**.

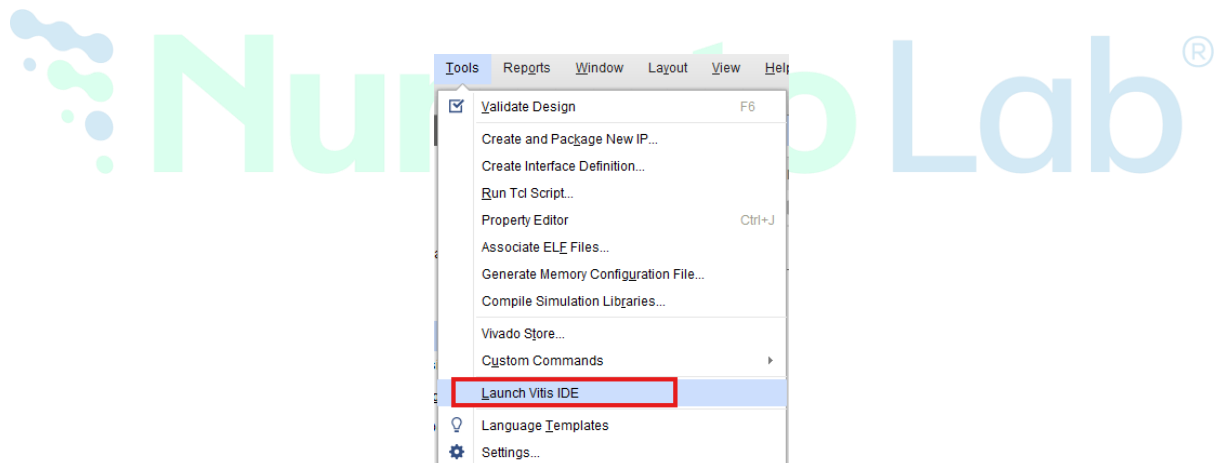


Select the “**include bitstream**” checkbox and click **Next**.

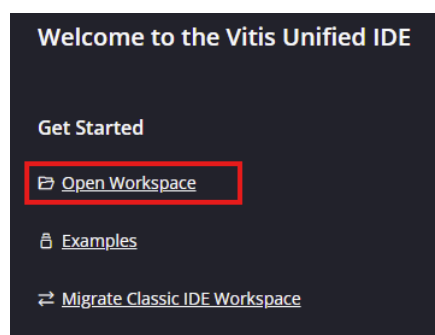




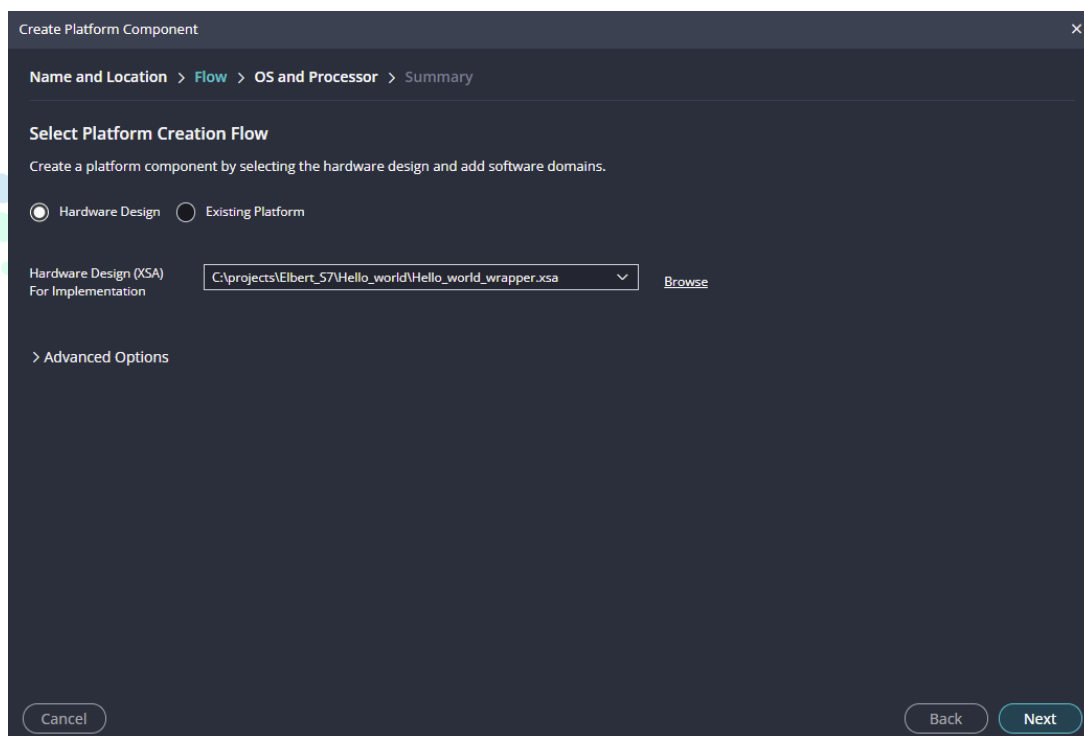
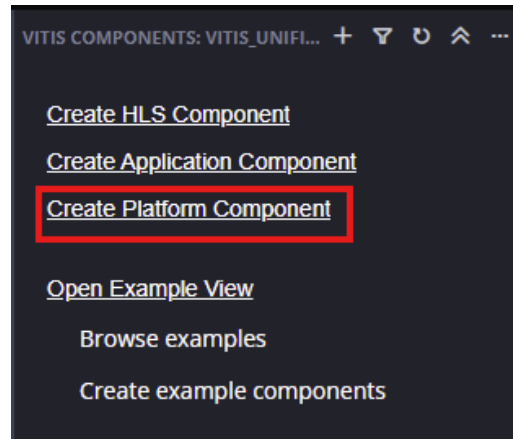
Step 13: Select **Launch Vitis IDE** from the **Tools** menu.



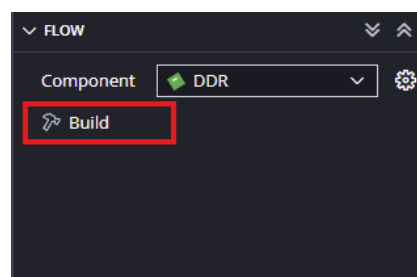
Step 14: After Vitis Unified IDE window opens, click on “**Open Workspace**” and select necessary folder to keep the Vitis files.



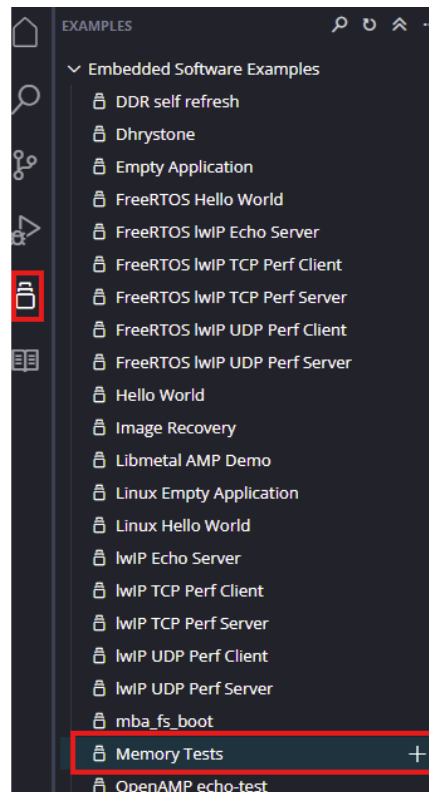
Step 15: Create a new platform for the project, by selecting “**Create Platform Component**”, click “**Next**”, in the Flow tab select the XSA file saved using the step 12 and finally click “**Next**” and “**Finish**” respectively.



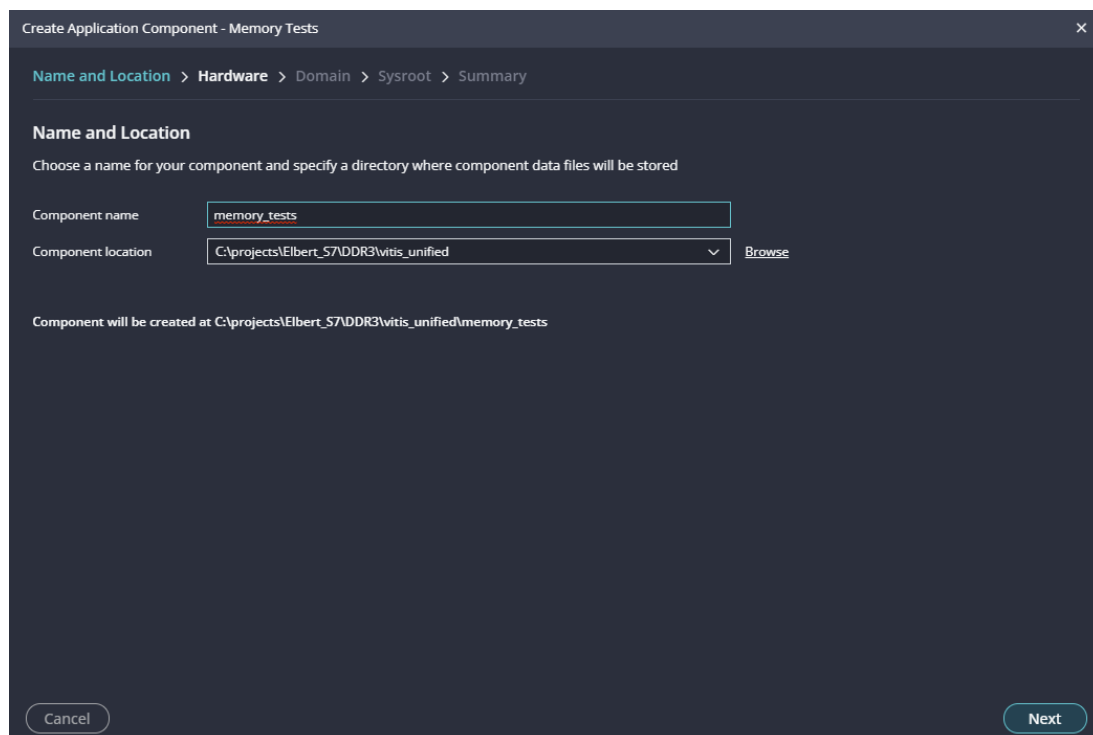
After successful creation of the platform, build the platform.



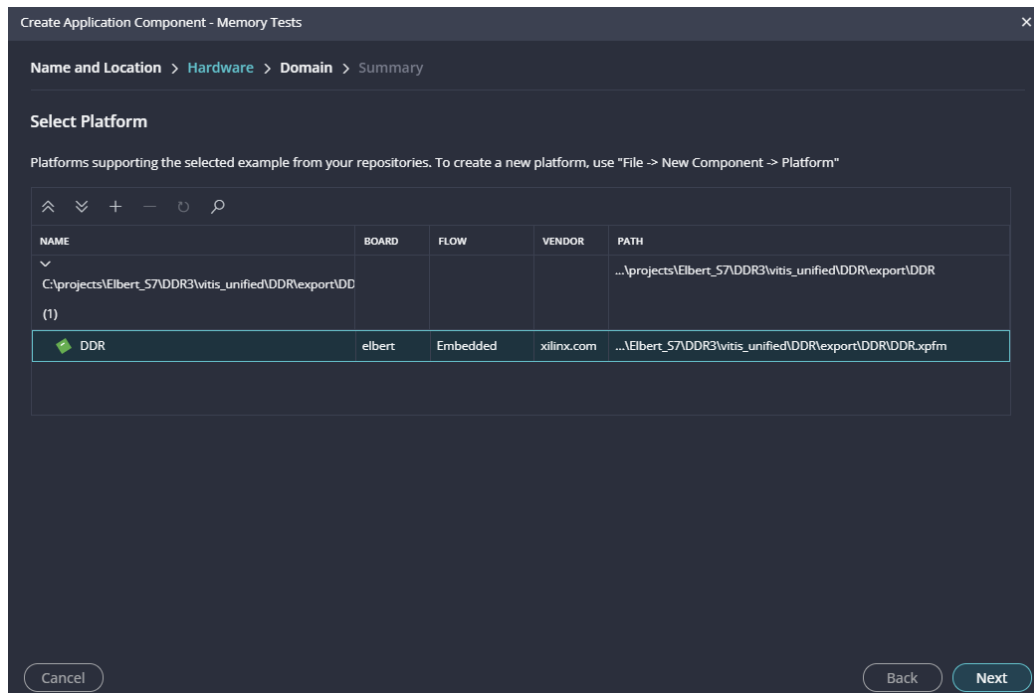
Step 16: Next create the DDR3_test Application component by selecting the “Memory tests” template from the “examples”,



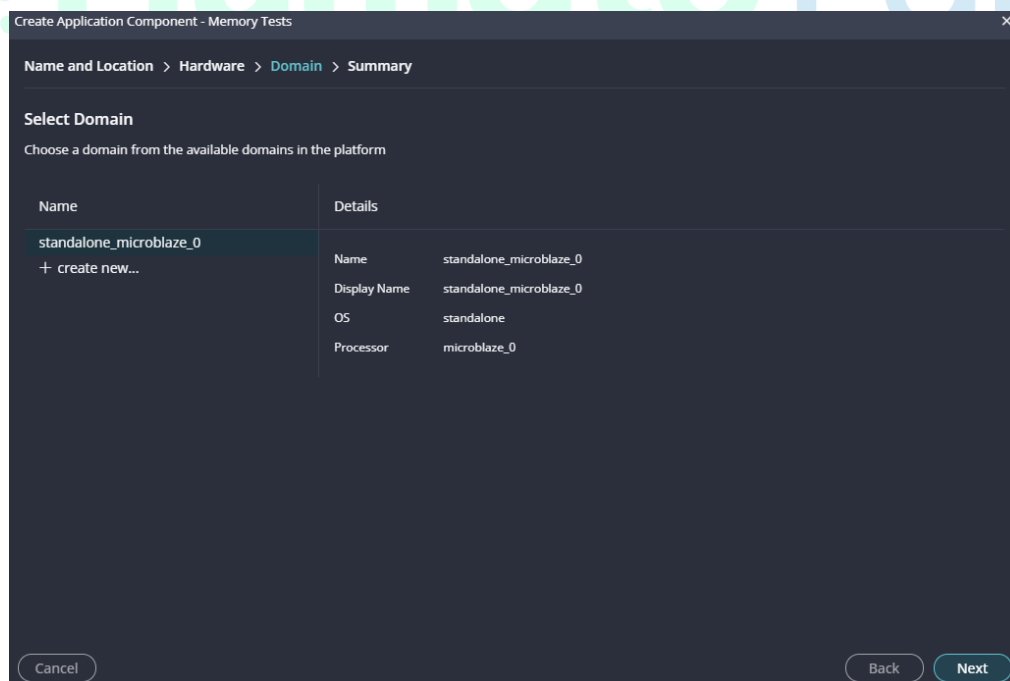
In “Create Application Component” tab specify project name and location, click “Next”



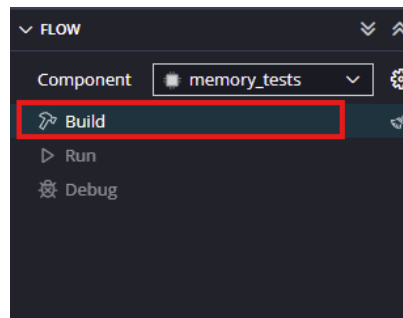
Select newly created Platform and click “Next”.



Select the domain as “**Standalone_microblaze_o**” and click “Next” and click on “Finish”

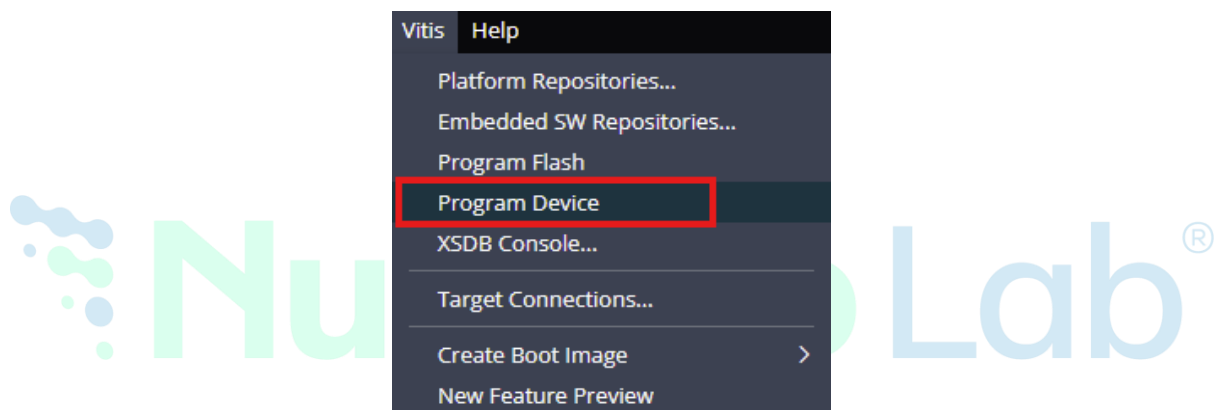


When the Memory tests project is added successfully, build the project manually.

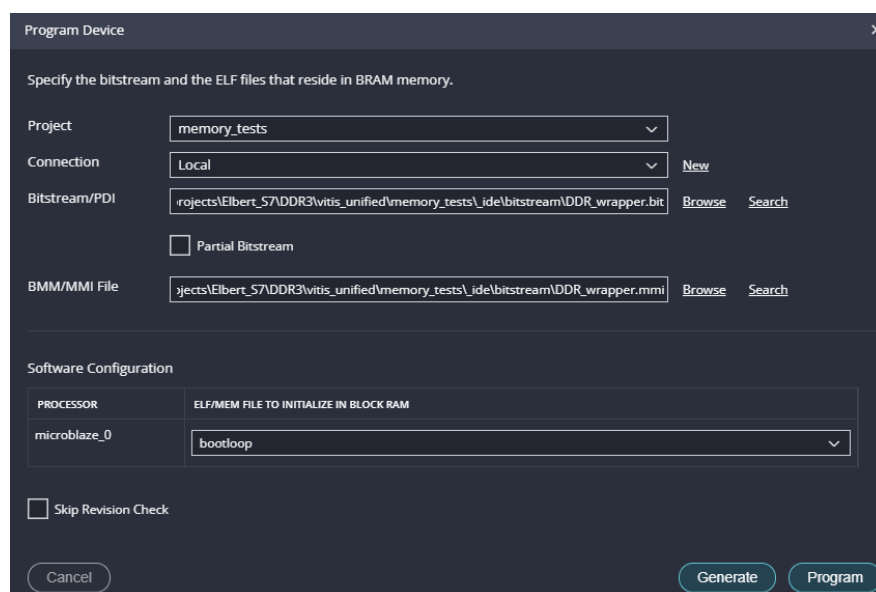


Step 17: Once the build is completed successfully, power up Elbert S7 using an USB type C cable

Step 18: Program the FPGA on Elbert S7 with a simple boot loop program by selecting the **Program Device** option from the **Vitis menu**.

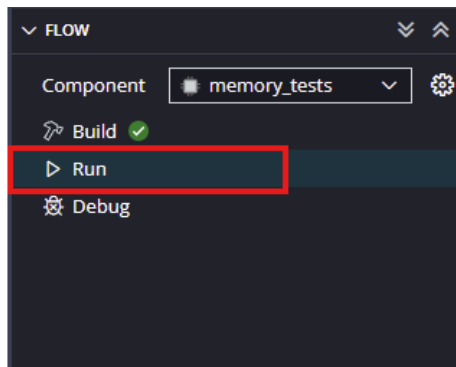


Once the “**Program Device**” window opens click on “**Program**”.

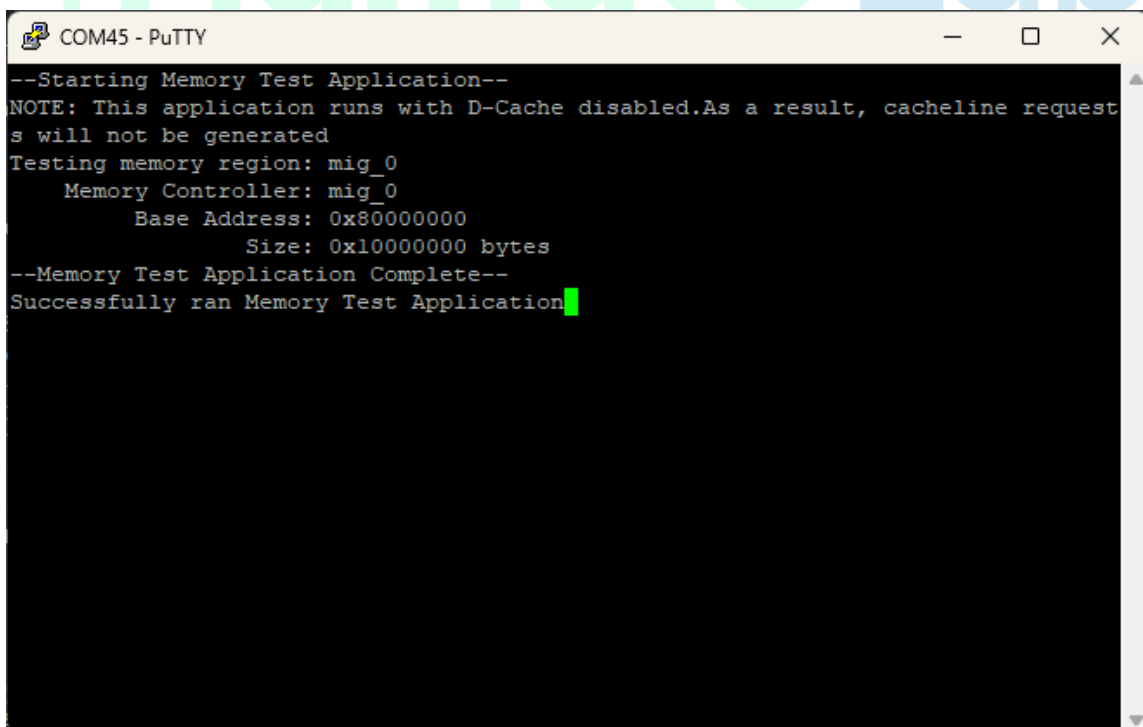


Step 19:

Meanwhile, open any serial terminal program (such as PuTTY, Teraterm etc) and open the port corresponding to Elbert S7 with a 9600 baud rate (the default baud rate given in UART IP). Program the board by selecting the “Run”.

**Step 20:**

If everything went well, the application running on the board should print the memory testing Process over the UART and should be displayed on the Serial Terminal application.

A screenshot of a PuTTY serial terminal window titled 'COM45 - PuTTY'. The terminal displays the following text:

```
--Starting Memory Test Application--  
NOTE: This application runs with D-Cache disabled.As a result, cacheline request  
s will not be generated  
Testing memory region: mig_0  
    Memory Controller: mig_0  
        Base Address: 0x80000000  
            Size: 0x10000000 bytes  
--Memory Test Application Complete--  
Successfully ran Memory Test Application
```

5. Gigabit Ethernet Example Design

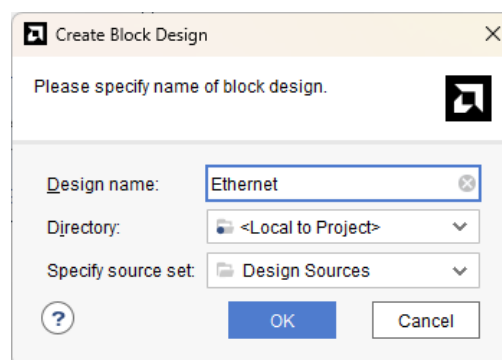
INTRODUCTION

Ethernet is a Link Layer Protocol in the TCP/IP protocol stack between the physical and data link layer. It is the most widely used protocol for Local Area Networks (LANs). Every device on Ethernet is assigned a unique MAC address for communication. [Gigabit Ethernet](#) refers to various technologies developed for transmitting Ethernet frames at the rate of gigabits per second. The [Reduced Gigabit Media-Independent Interface \(RGMII\)](#) is used to interface the Ethernet IP core on FPGA with the Gigabit Ethernet PHY chip on Elbert S7. The Media Access Layer converts the packets into a stream of data to be sent while the Physical Layer converts the stream of data into electrical signals. RGMII provides a media-independent interface so that MAC and PHY can be compatible, irrespective of the hardware used. In this tutorial, the Numato Lab Elbert S7 FPGA Development Board is used to demonstrate a TCP/IP echo server application. The echo server application runs on lightweight IP (lwIP) TCP/IP stack.

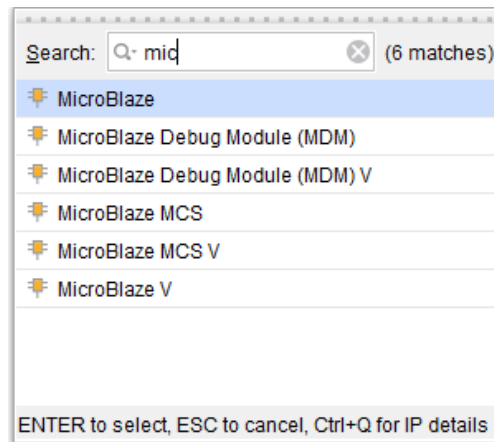
STEP 1:

To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section “[Getting Started with Vivado: Creating a New FPGA Project.](#)”

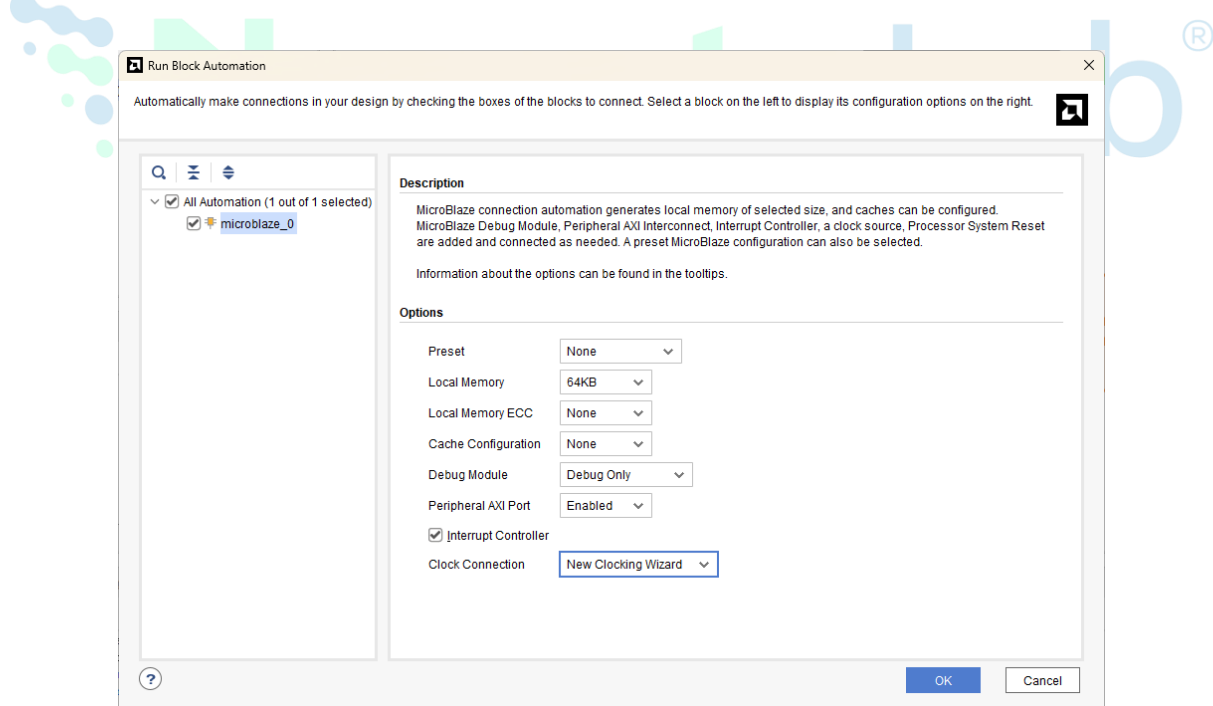
Step 2: After creating a new project successfully, In the Flow Navigator panel, select Create Block Design under IP INTEGRATOR. Enter a name for the block design and click OK. An empty block design will be created



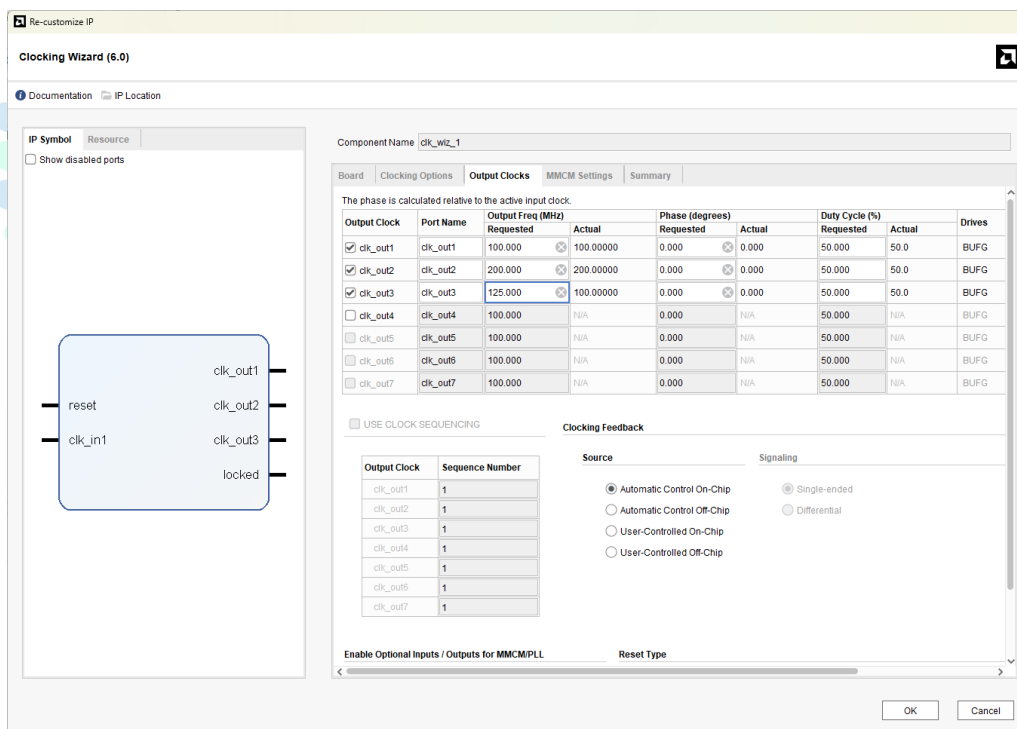
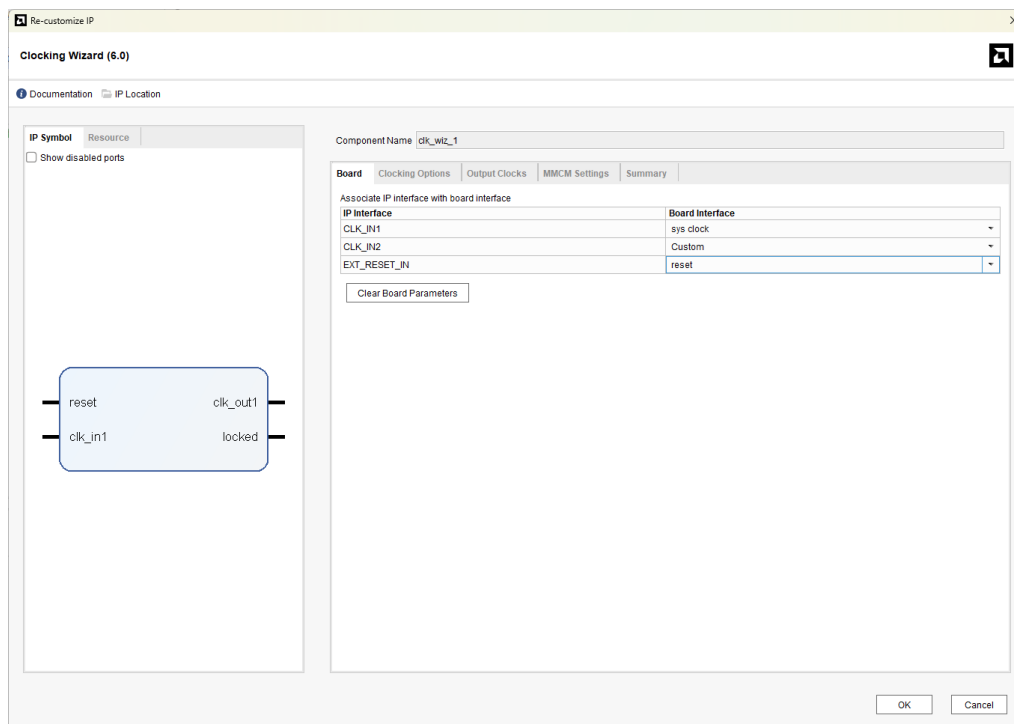
Step 3: Go to Diagram window, right click and select “Add IP” from the popup menu. Search for “MicroBlaze” and add it to the design by double-clicking it.



Click “Run Block Automation” present in the “Designer Assistance available” bar on the top left corner of the window to complete the design. Select the settings as shown in the following image. Click “OK” for Vivado to automatically configure the blocks for you.

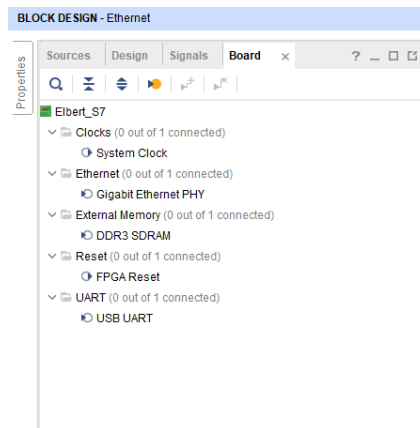


Step 4: Double click “Clocking Wizard” IP and customize “Board” settings as shown in the following image.



Click **OK** to customize the IP.

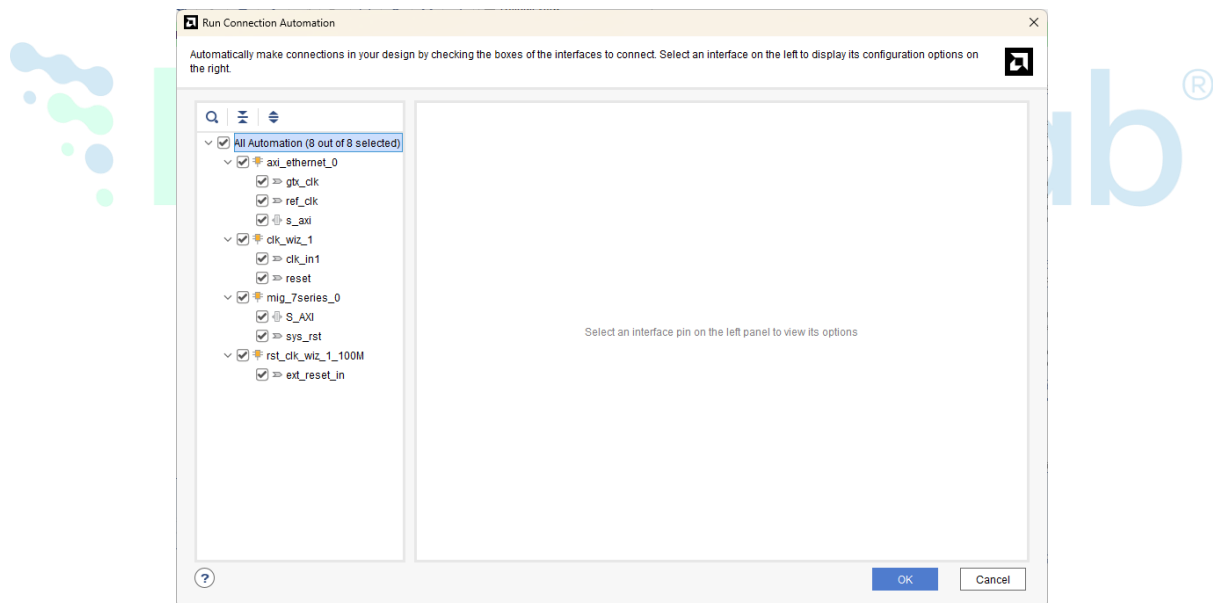
Step 5: Click the **Board** tab. The default peripherals available for the Elbert S7 board will be displayed.



Drag and drop **DDR3 SDRAM**, **USB UART**, and **Gigabit Ethernet PHY** into IP Canvas.

Ensure that **sys_clk_i** of Memory Interface Generator is connected to **clk_out2**.

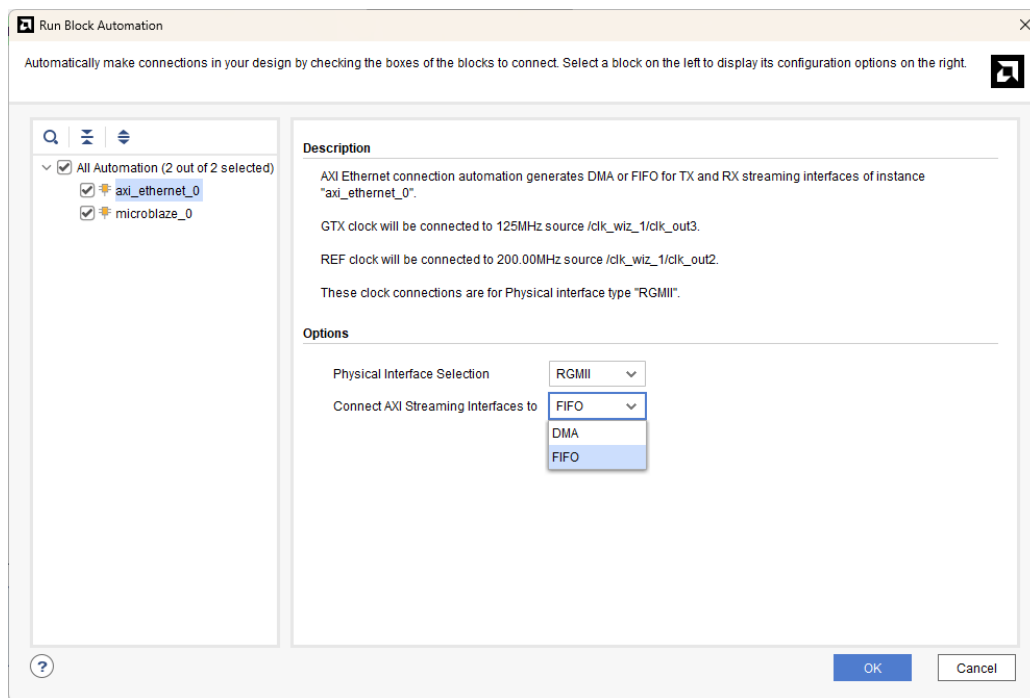
Step 6: Click **Run Connection Automation** and select all.



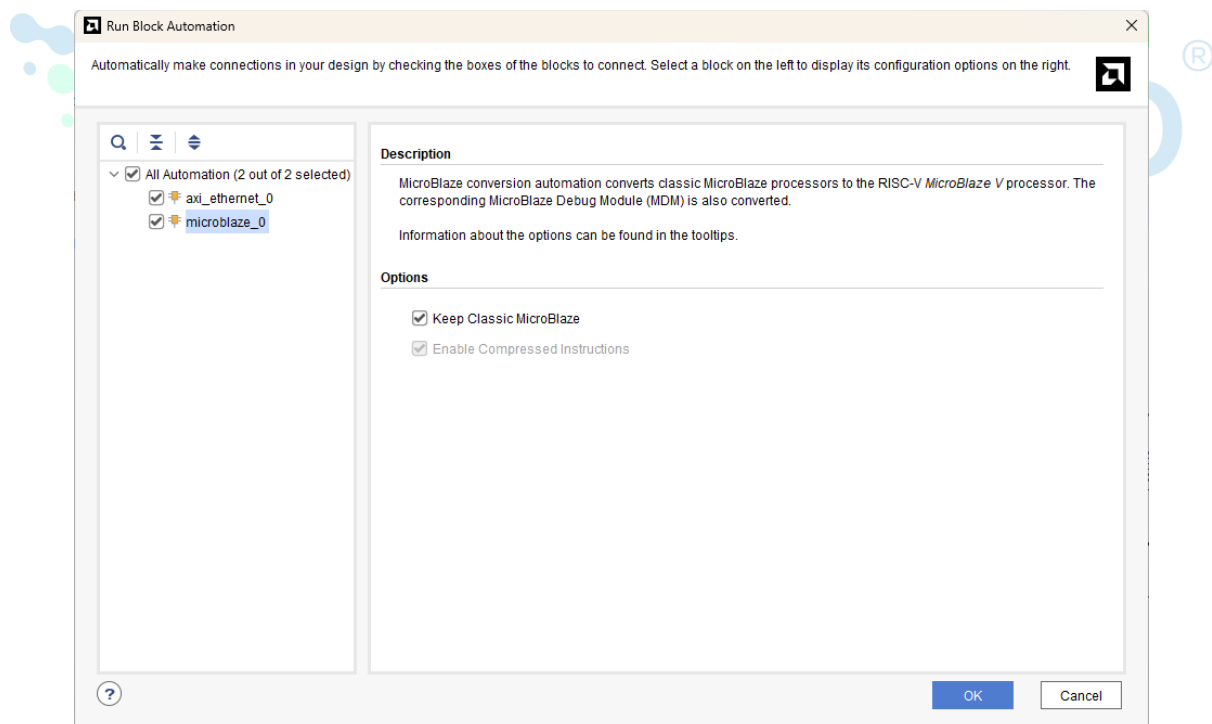
Step 7:

Run Block Automation

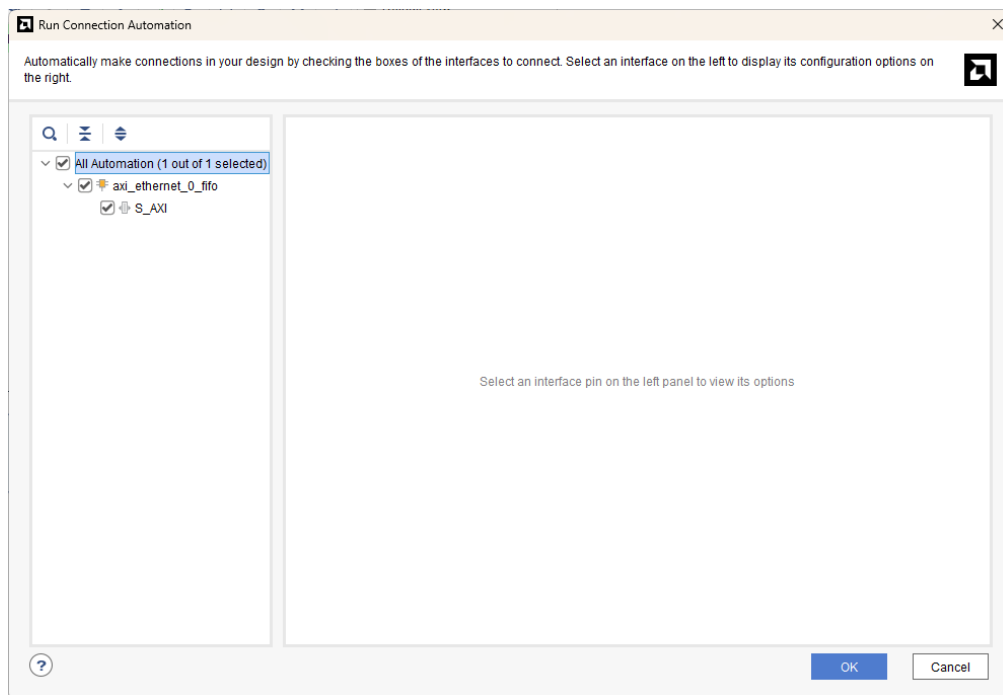
- for AXI Ethernet and select **"FIFO"** for the AXI Streaming interface.



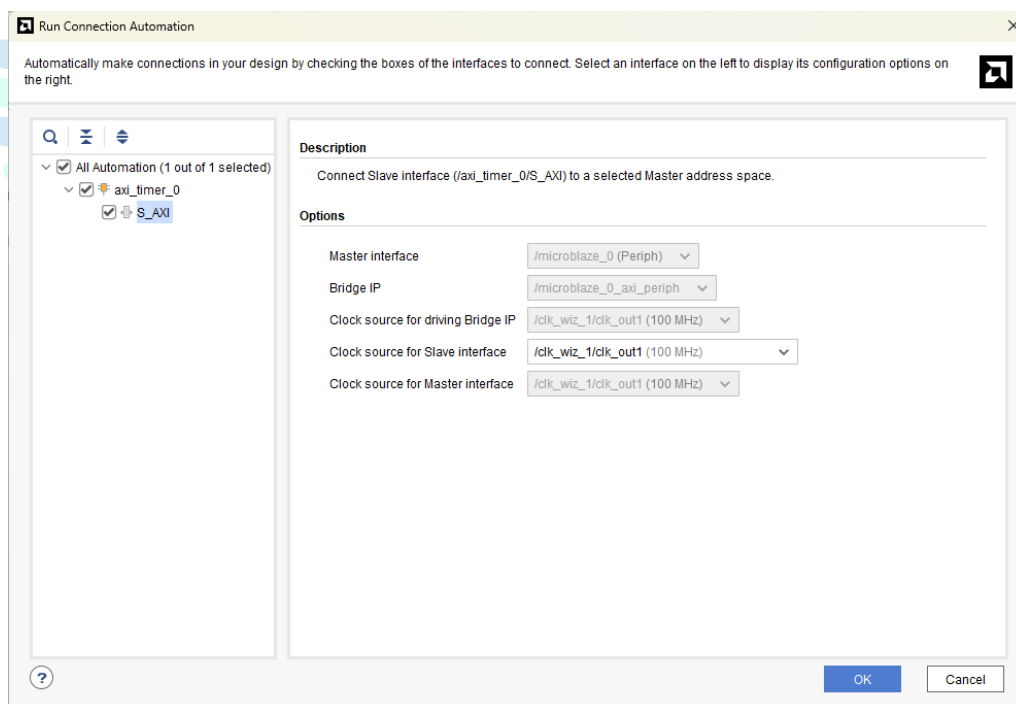
- for Microblaze_0 select "**Keep Classic MicroBlaze**" option and click "OK"



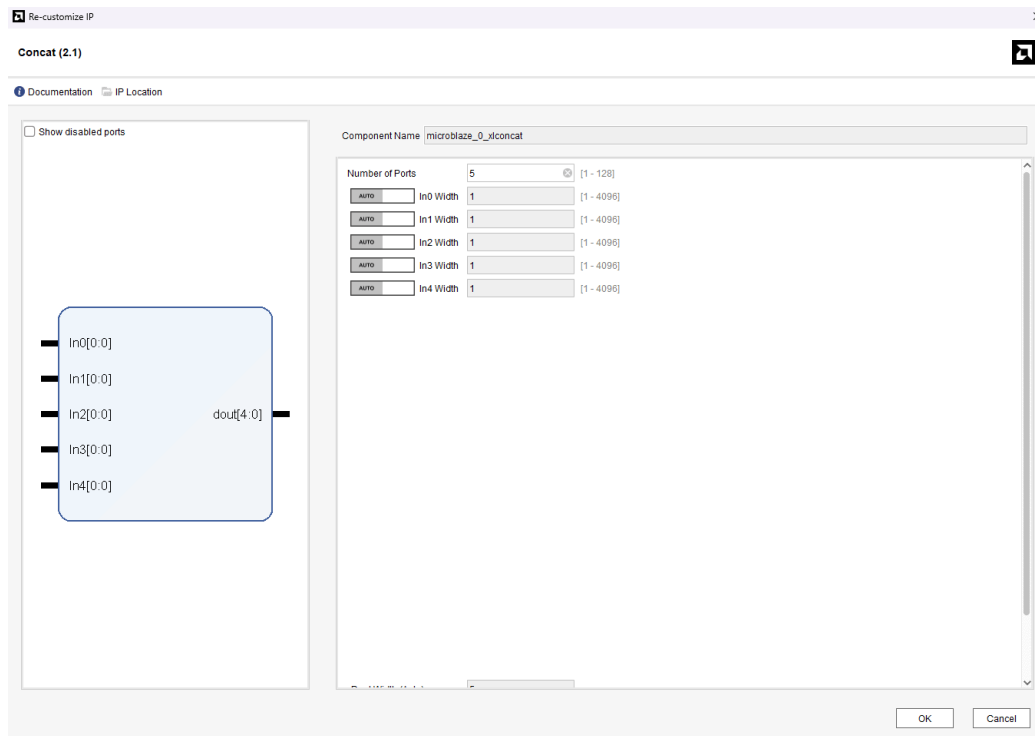
Step 8: Click **Run Connection Automation**. Select the **All Automation** option and click **OK**.



Step 9: Add AXI Timer into IP Canvas and click **Run Connection Automation.**

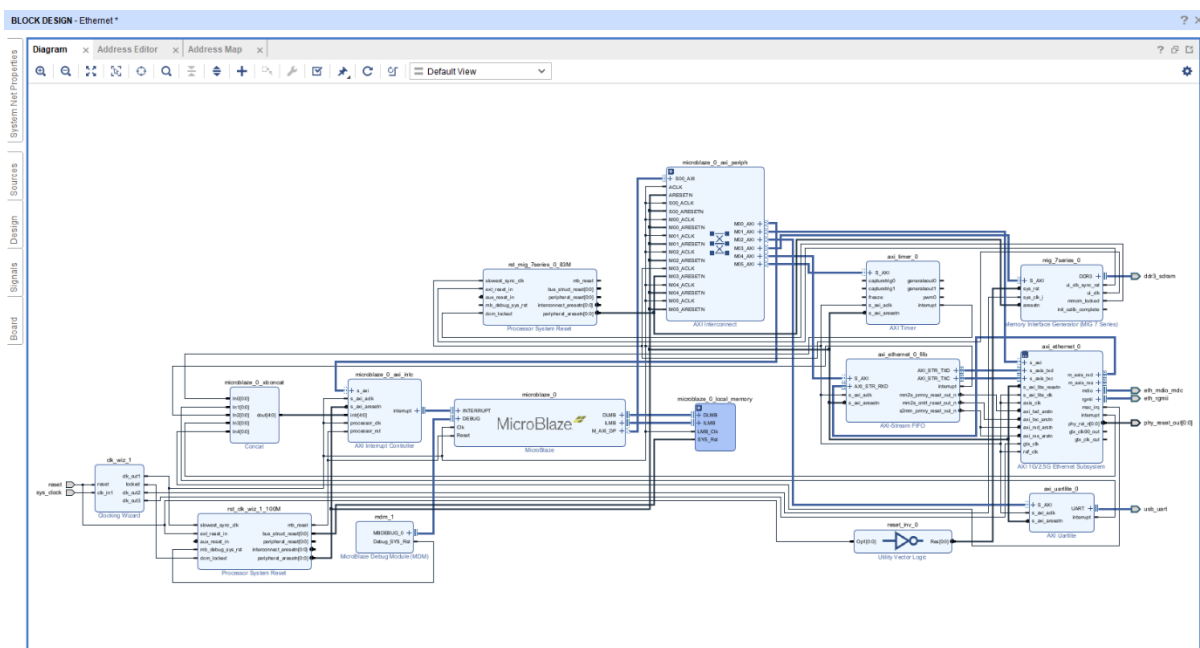


Step 10: Customize the Concat IP block as shown below.



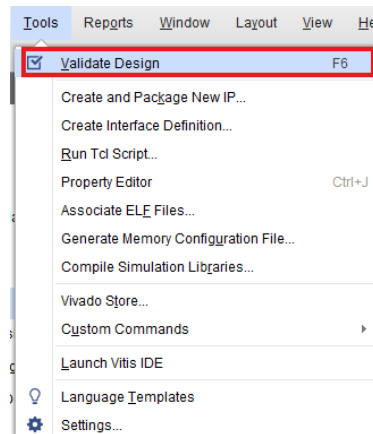
Route the following connections to the inputs of the Concat block:

- **interrupt** on AXI Uartlite block
- **interrupt** on AXI Timer block
- **interrupt** on AXI-Stream FIFO
- **interrupt** and **mac_irq** on AXI 1G/2.5G Ethernet Subsystem

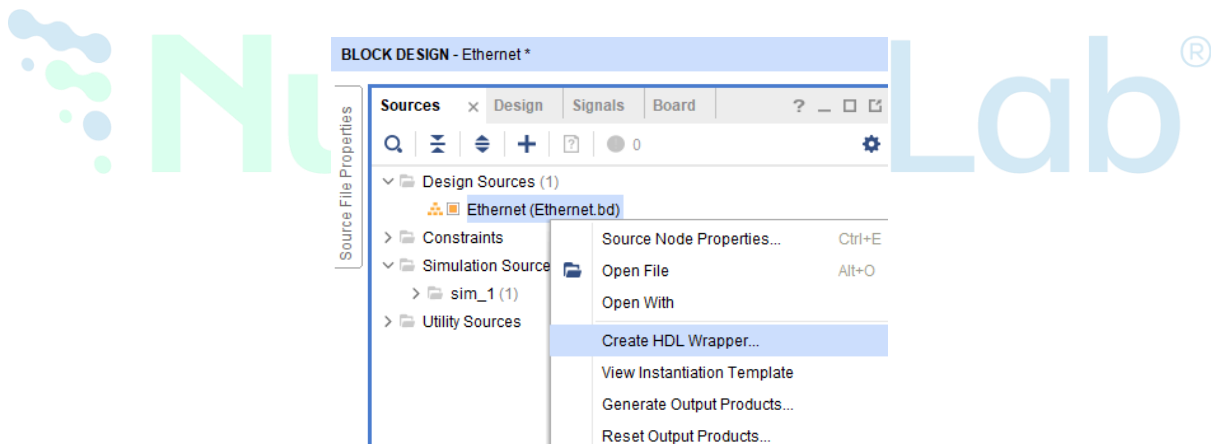


Make sure that the final design looks as shown above.

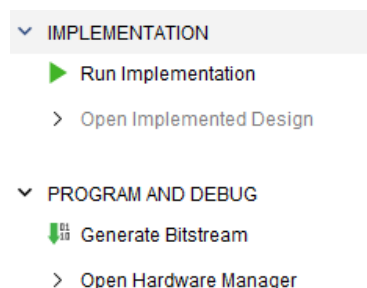
Step 11: Select the **Validate Design** option from the **Tools** menu to ensure that connections are correct.



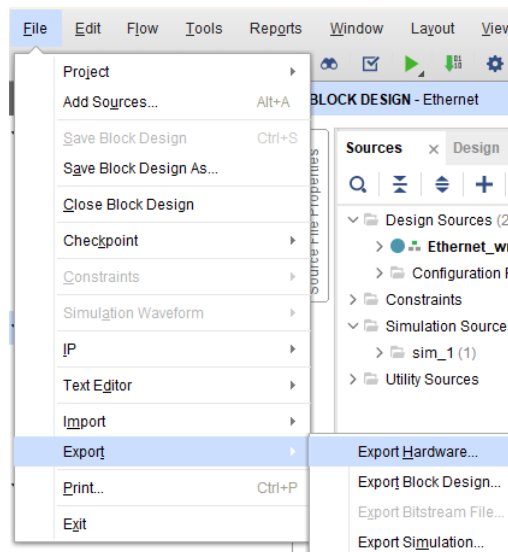
Step 12: In the **Sources** window, right-click on the design and select **Create HDL Wrapper**. Click **OK** in the dialog box that appears.



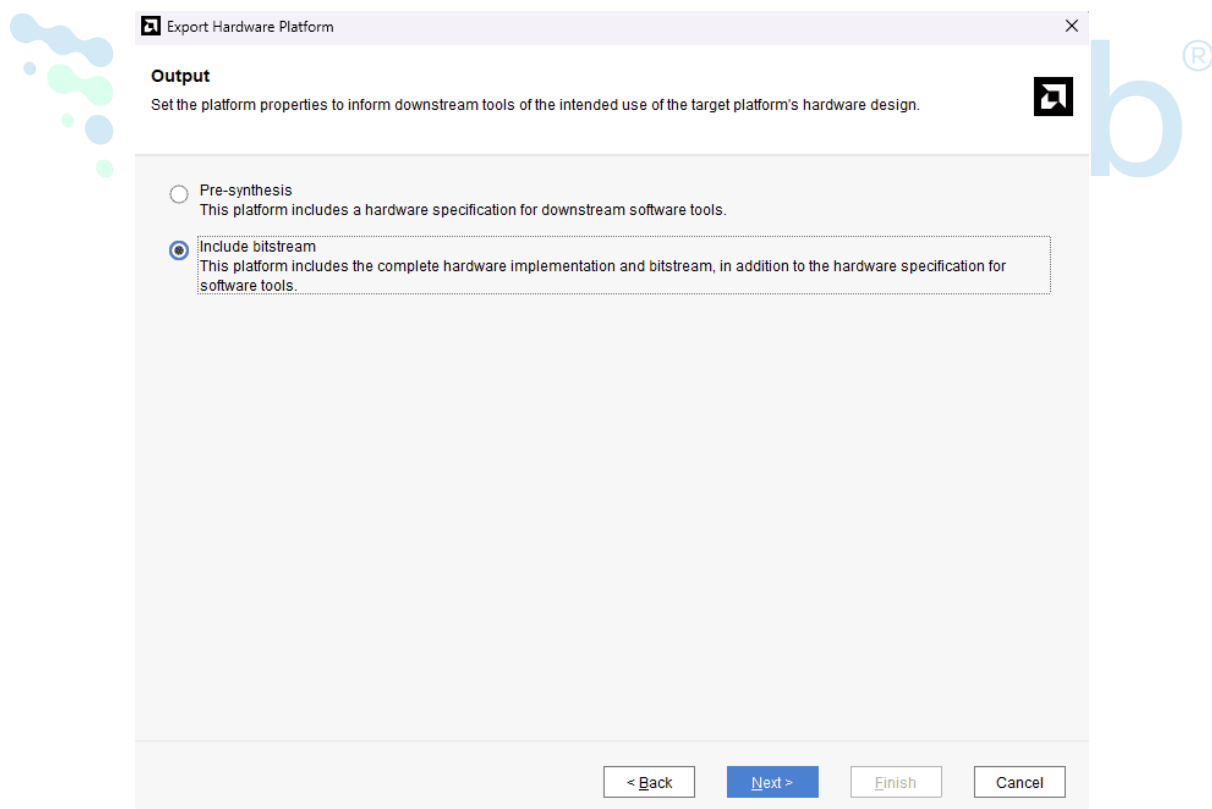
Step 13: Click **Generate Bitstream** under the **PROGRAM AND DEBUG** section of Vivado to synthesize, implement and generate the bitstream.



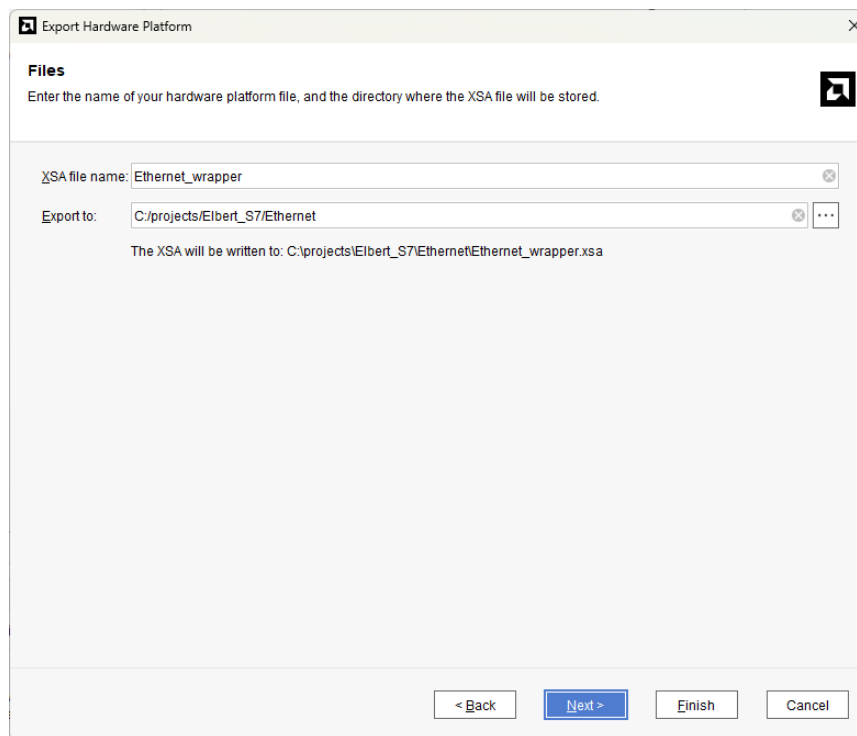
Step 14: After generating the bitstream successfully, select **Export -> Export Hardware** from the **File menu**. Click **Next**.



Select the “**include bitstream**” checkbox and click **Next**.



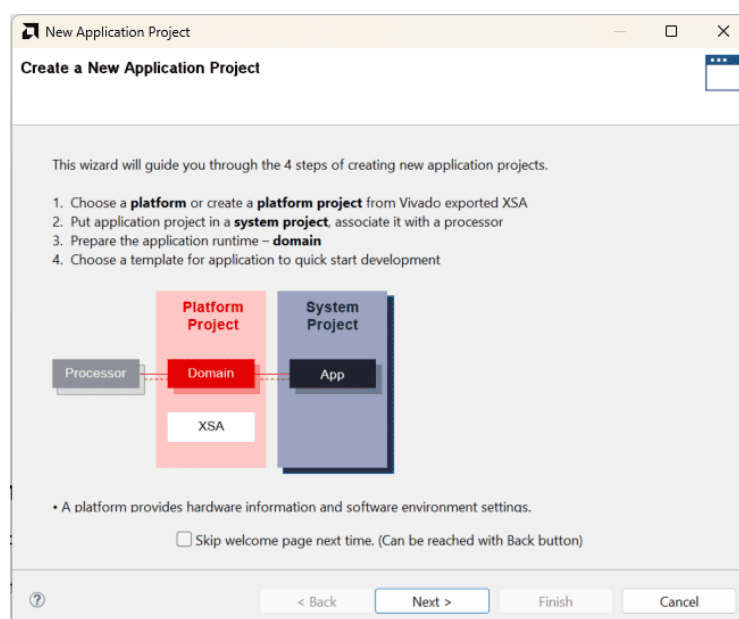
Provide the **XSA file name** and save it at a suitable **location**. Click **Next** and click **Finish** in the next dialog box.



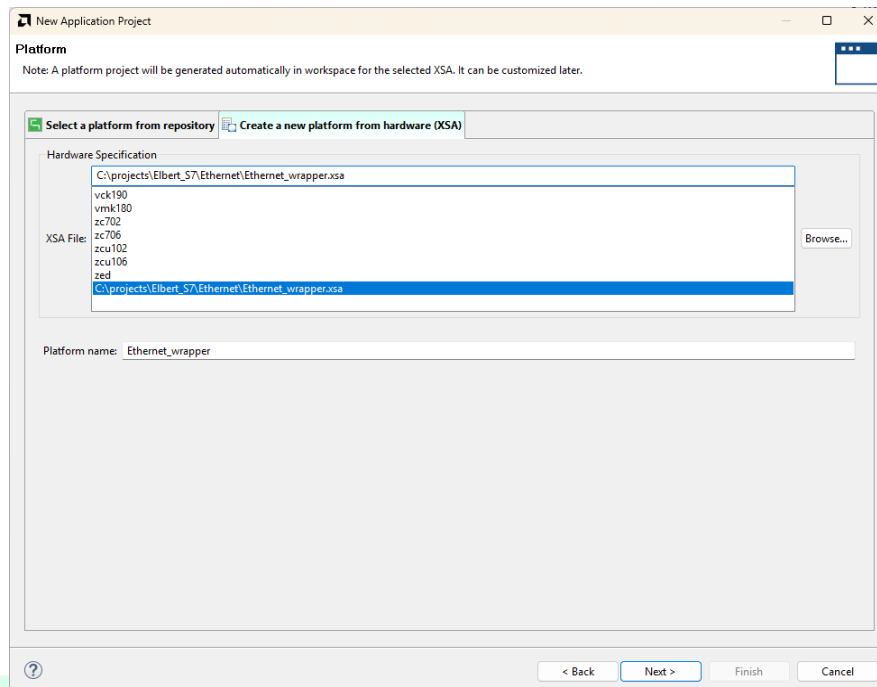
Step 15: Launch Vitis classic.

Note: In Vivado 2024.1, accessing Vitis via the tools menu inadvertently launches Vitis Unified instead of Vitis Classic, which is our preferred tool for project creation. To utilize Vitis Classic, it is necessary to launch it separately.

Step 16: In Vitis, IDE window select **Create Application Project** and click **Next** in the dialog box that appears.

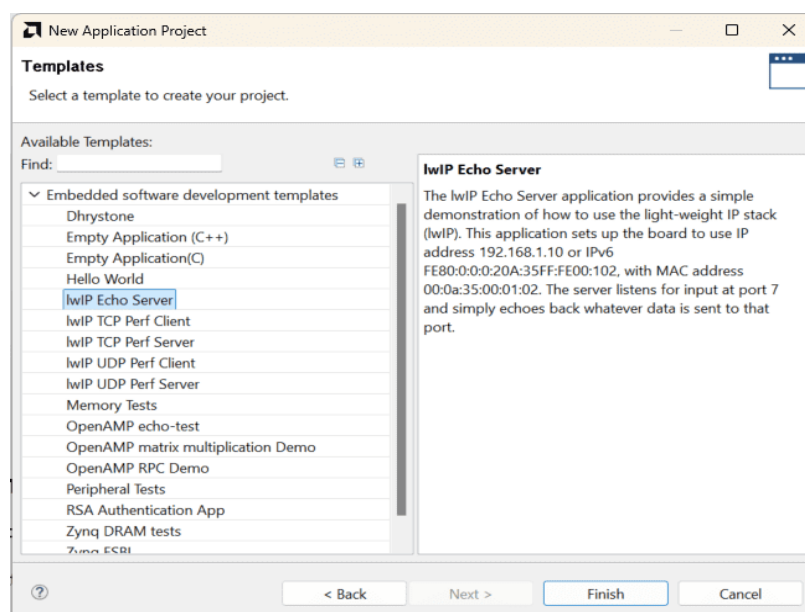


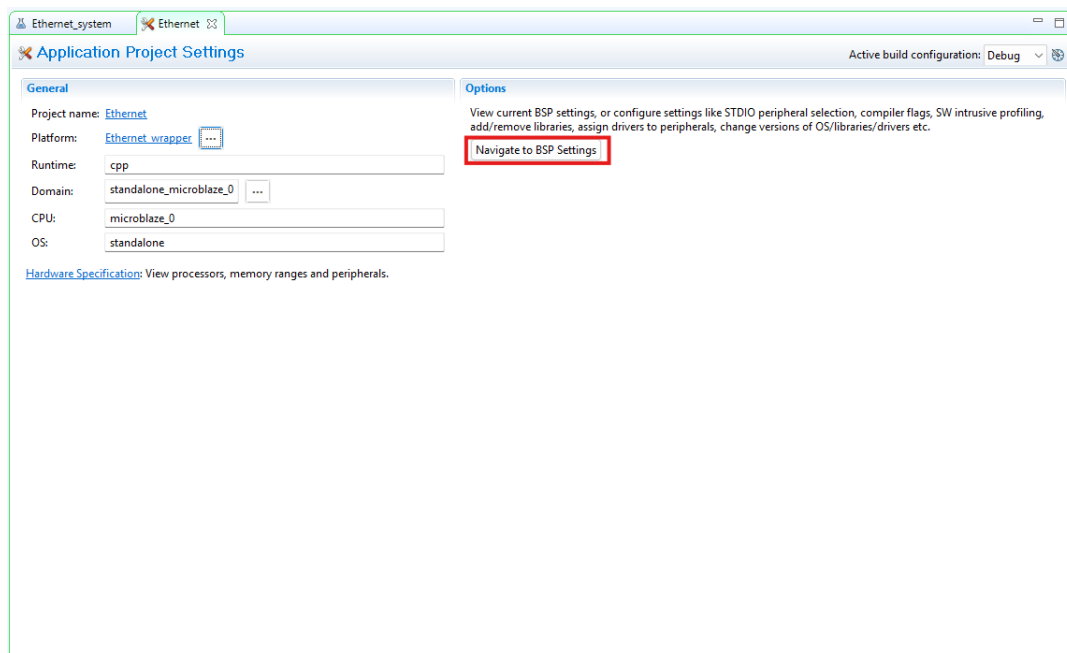
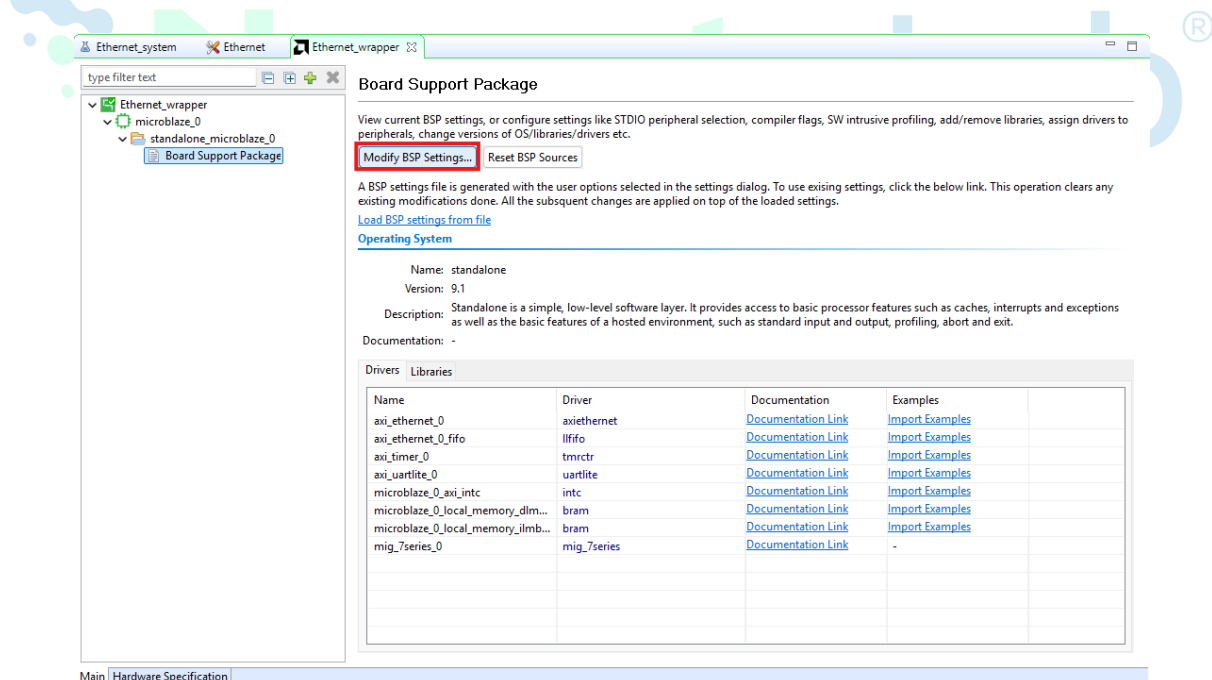
In the **Platform**, window select **Create a new platform from the hardware** Tab and import the **XSA file** which is already created (Provide XSA file location). Click **Next**.



In the **Application Project Details** window, give an appropriate **name** for the Vitis Project and click **Next**. Click **Next** in the **Domain** window.

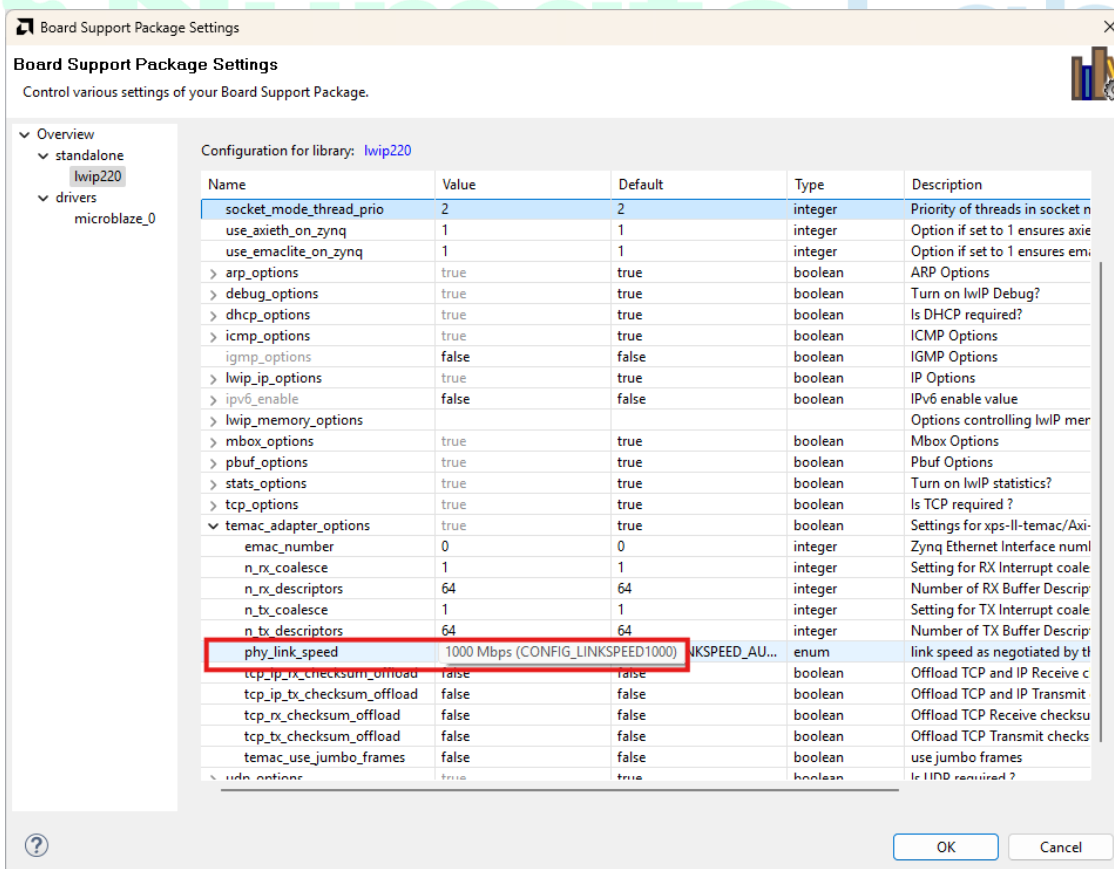
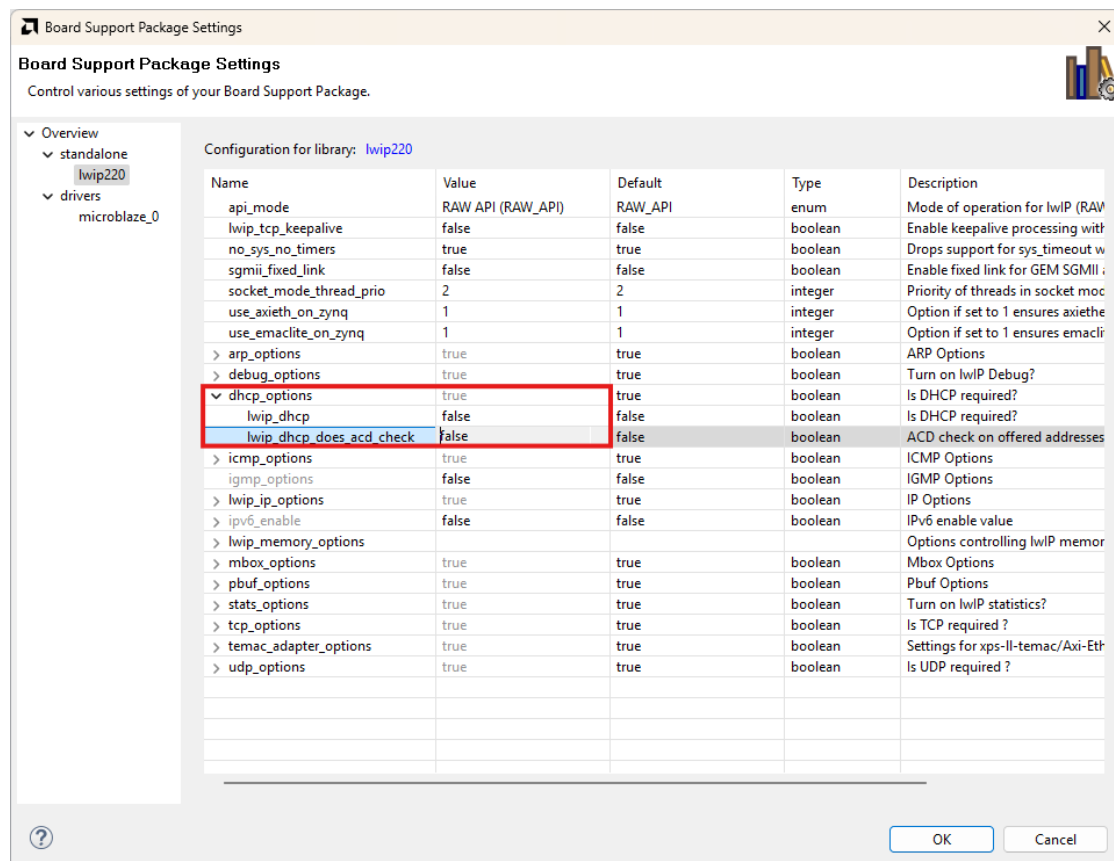
Select the **lwIP Echo Server** template from the list of available templates and click **Finish**.



Step 17: Select Navigate to BSP Settings from Application Project Settings.**Select Board Support Package** and click on **Modify BSP Settings** option.

In the Board Support Package Settings window, select lwip (Iwip220) library, change the **dhcp_options** to “false” and ensure that “debug options” are “false”.

Select **phy_link_speed** in **temac_adapter_options** as **CONFIG_LINKSPEED1000**.



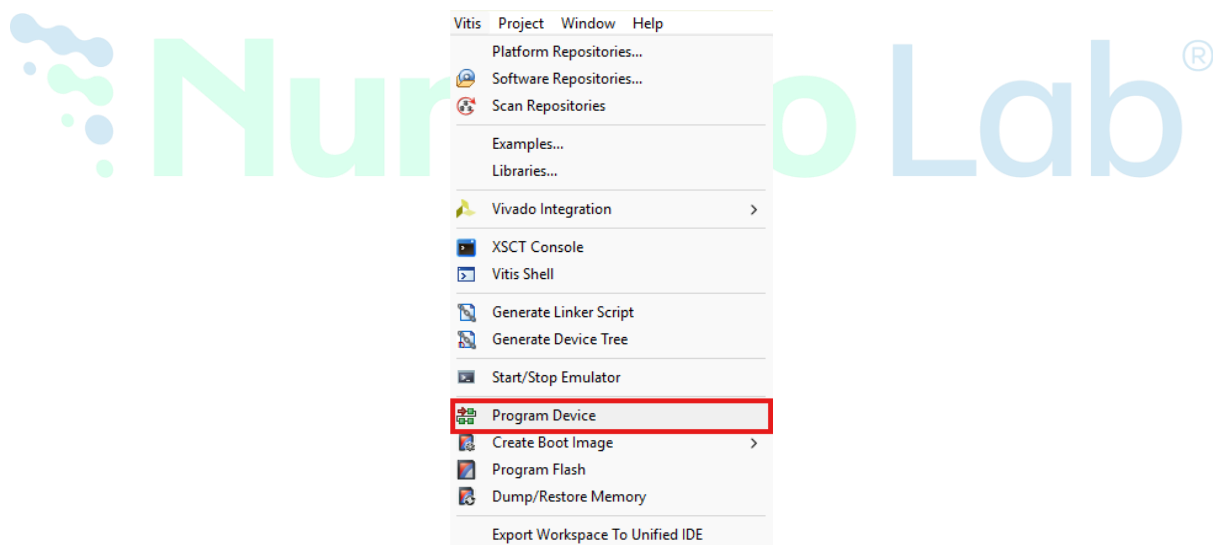
After changing the library settings, click OK. vitis will update the BSP automatically. If that didn't happen for any reason, run a build manually.

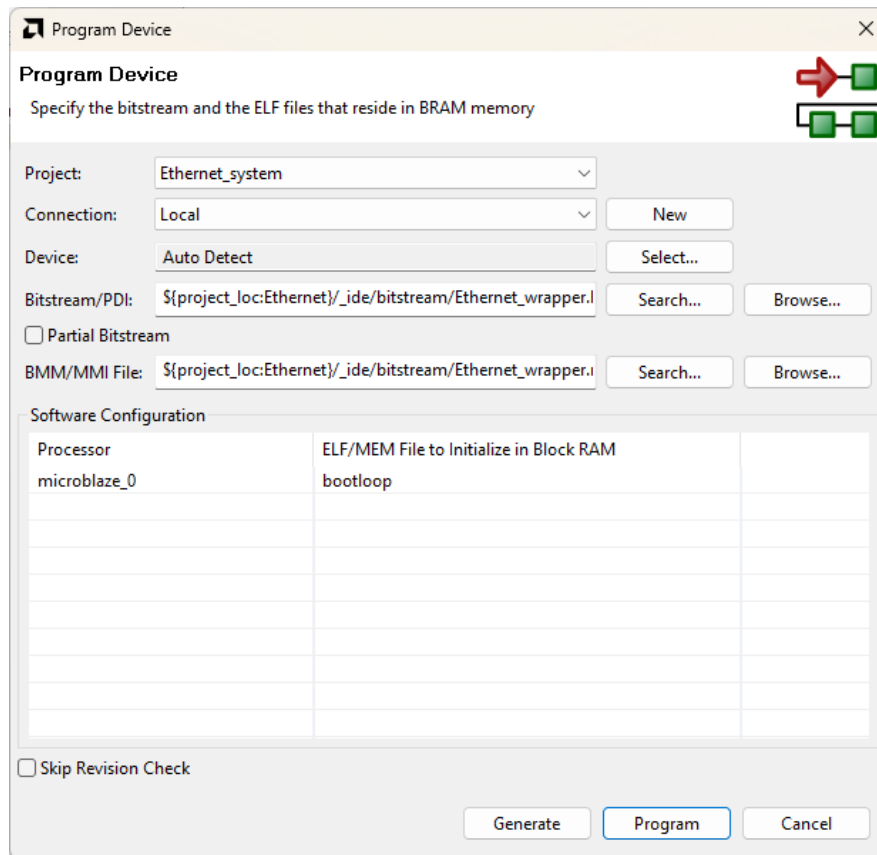
NOTE: Vitis does not include built-in support for KSZ Ethernet PHY drivers. To enable compatibility, the xaxiemicif_physpeed file must be manually updated with the KSZ driver modifications. Replace the existing xaxiemicif_physpeed file in your project with the provided [file](#), which includes the necessary changes to support KSZ PHY drivers. This ensures proper Ethernet functionality in your application.

After modifying the xaxiemicif_physpeed file Build the project.

Step 18: Once the build is completed successfully, power up Elbert S7 using an USB type C cable.

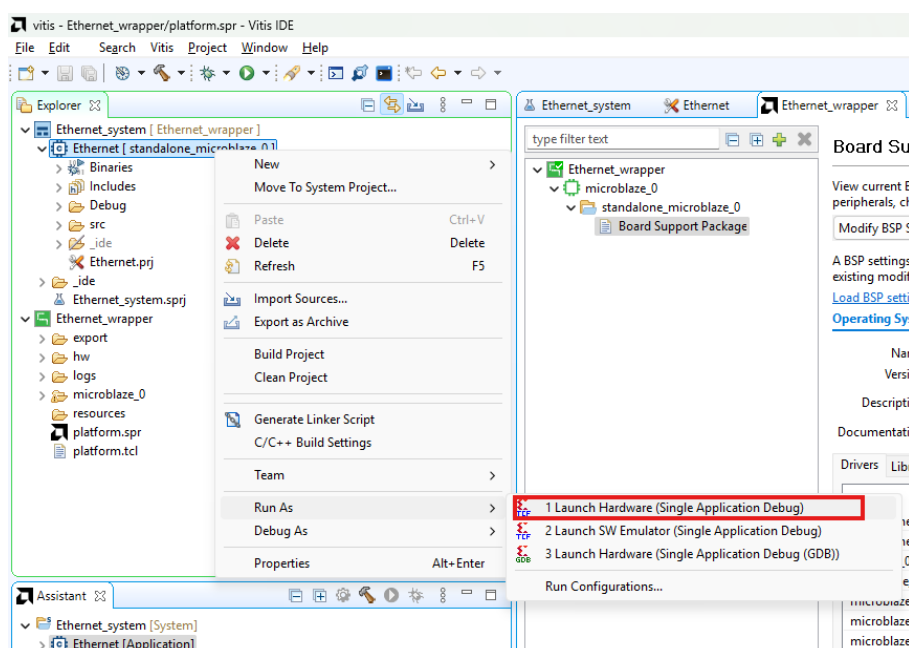
Step 19: Program the FPGA on Elbert S7 by selecting the **Program Device** option from the **vitis** menu.



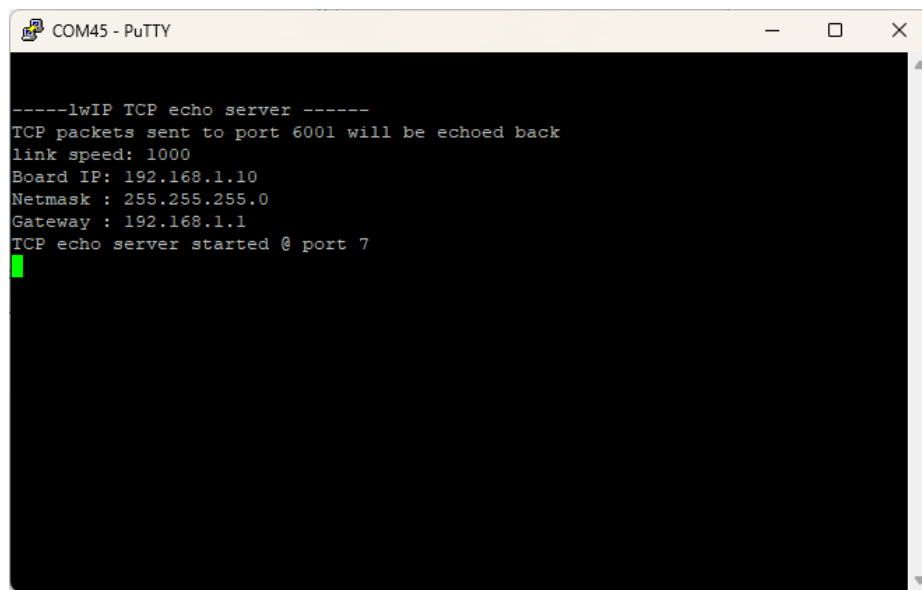


Step 20:

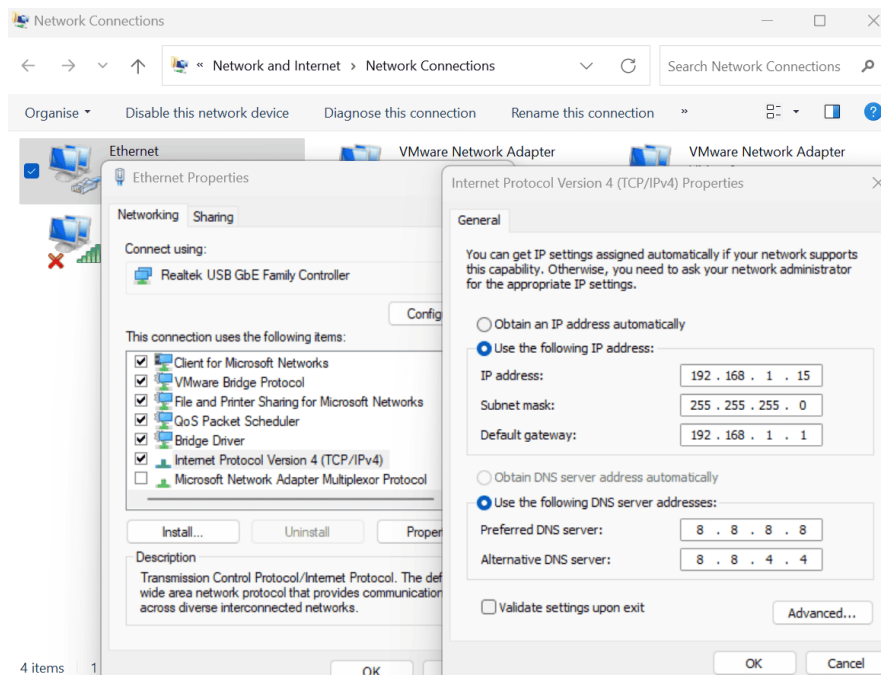
Open the COM port corresponding to Elbert S7 in any serial terminal (PuTTY, Tera Term, etc.) with a 9600 baud rate. Now, right-click on the .elf file in Project Explorer and select “Launch on Hardware” as shown below.



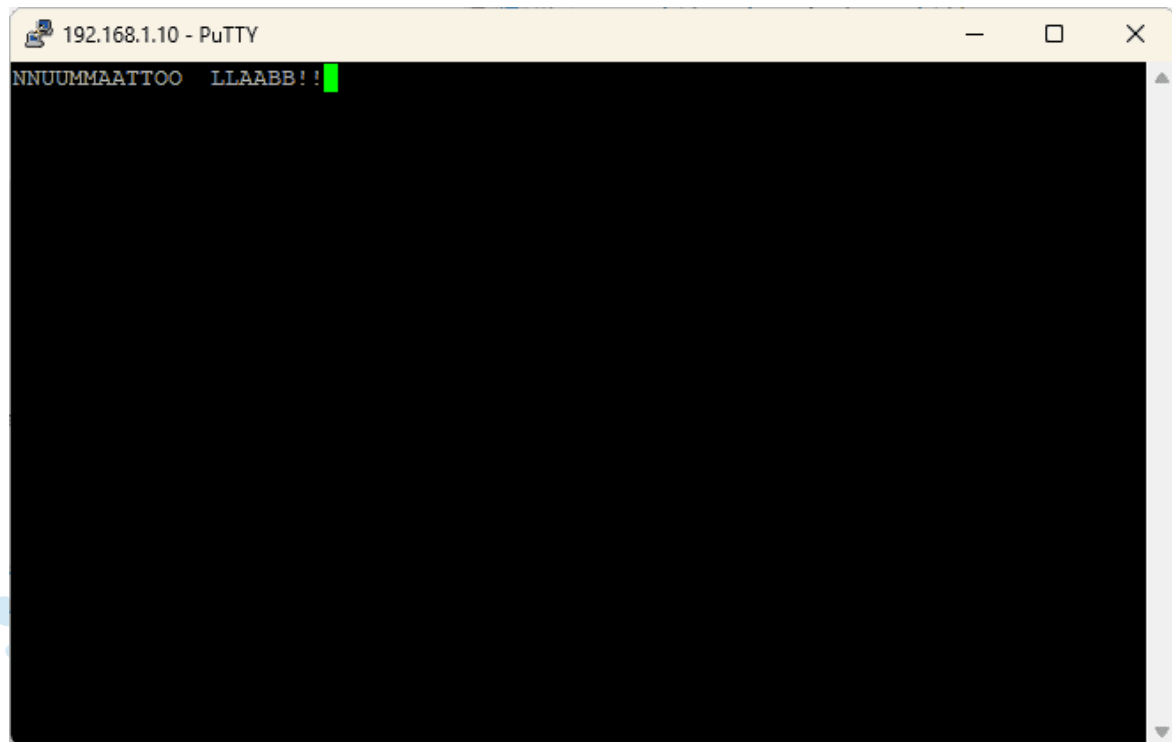
Observe the details displayed on the serial terminal.



Step 21: Connect the Ethernet cable to the board and the other end to the PC Ethernet port. Go to **Control Panel**. Go to **Network and Internet -> Network and Sharing Centre -> Change adapter settings**. Select “Change adapter settings”. Right-click on Ethernet, click properties, and select “IPv4”. Change the IPv4 address to **192.168.1.15** (any IP address can be used) and the default gateway to **192.168.1.1**.



Step 22: Open a telnet session with IP Address **192.168.1.10** (IP address as per main.c) at **port 7**, give input through the keyboard and observe the output. If you enter a character from the keyboard, you can observe the transmitted and echoed characters on telnet as shown.



6. SD card test

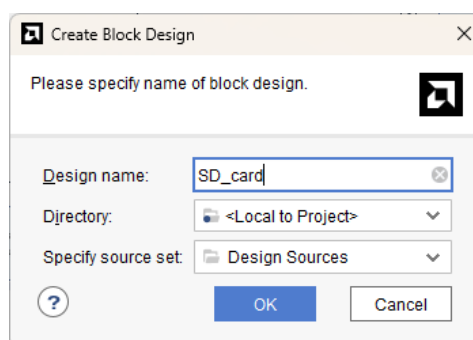
Introduction

In this tutorial, we'll explore the process of creating an SD card test project using Vivado and Vitis Unified IDE for the Elbert S7 FPGA Development Board. Our design will feature a MicroBlaze soft processor, which will be integrated with an SD card interface via the AXI bus. This project will allow us to perform basic read and write operations on the SD card, helping to verify the functionality of the SD card interface on the Elbert S7. Although MicroBlaze designs can utilize either PLB or AXI bus systems, we'll focus on the AXI bus for this tutorial. For detailed information on MicroBlaze and additional resources, including the datasheet, please visit [AMD's dedicated MicroBlaze page](#).

STEP 1:

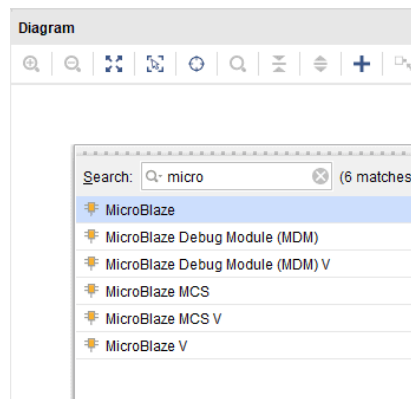
To create a new Vivado project specifically for the **Elbert S7** FPGA board, follow the procedure outlined in the section [“Getting Started with Vivado: Creating a New FPGA Project.”](#)

Step 2: After creating a project, In the “Flow Navigator” panel, select “Create Block Design” under the IP integrator section. Give an appropriate name (Eg: “SD_card”) to the design and click “OK”.

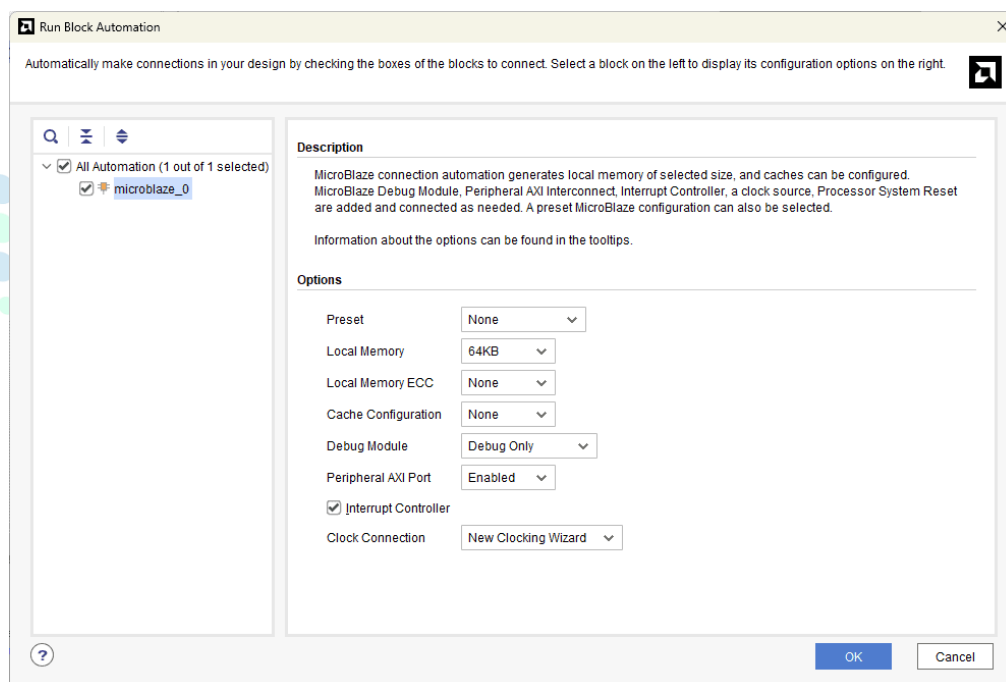


Step 3:

Go to Diagram window, right click and select “Add IP” from the popup menu. Search for “**MicroBlaze**” and add it to the design by double-clicking it.

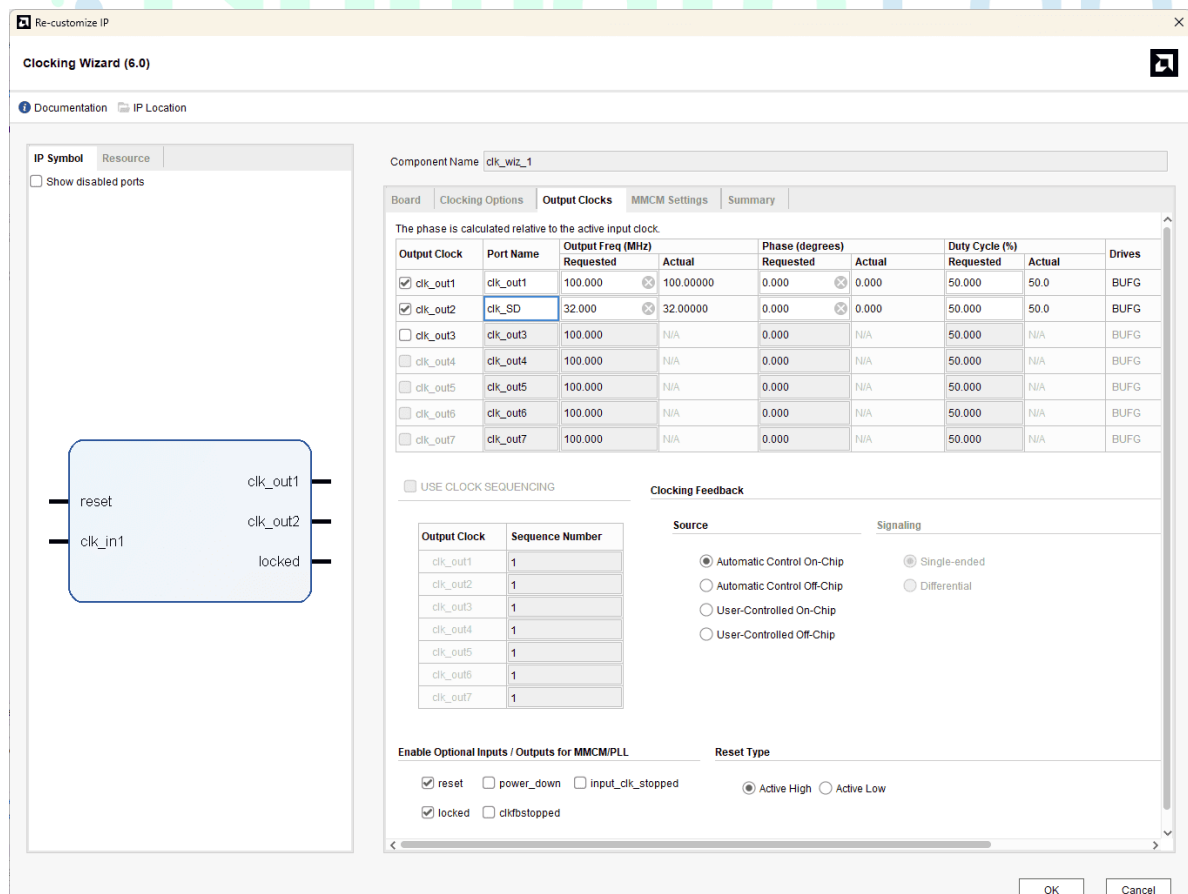
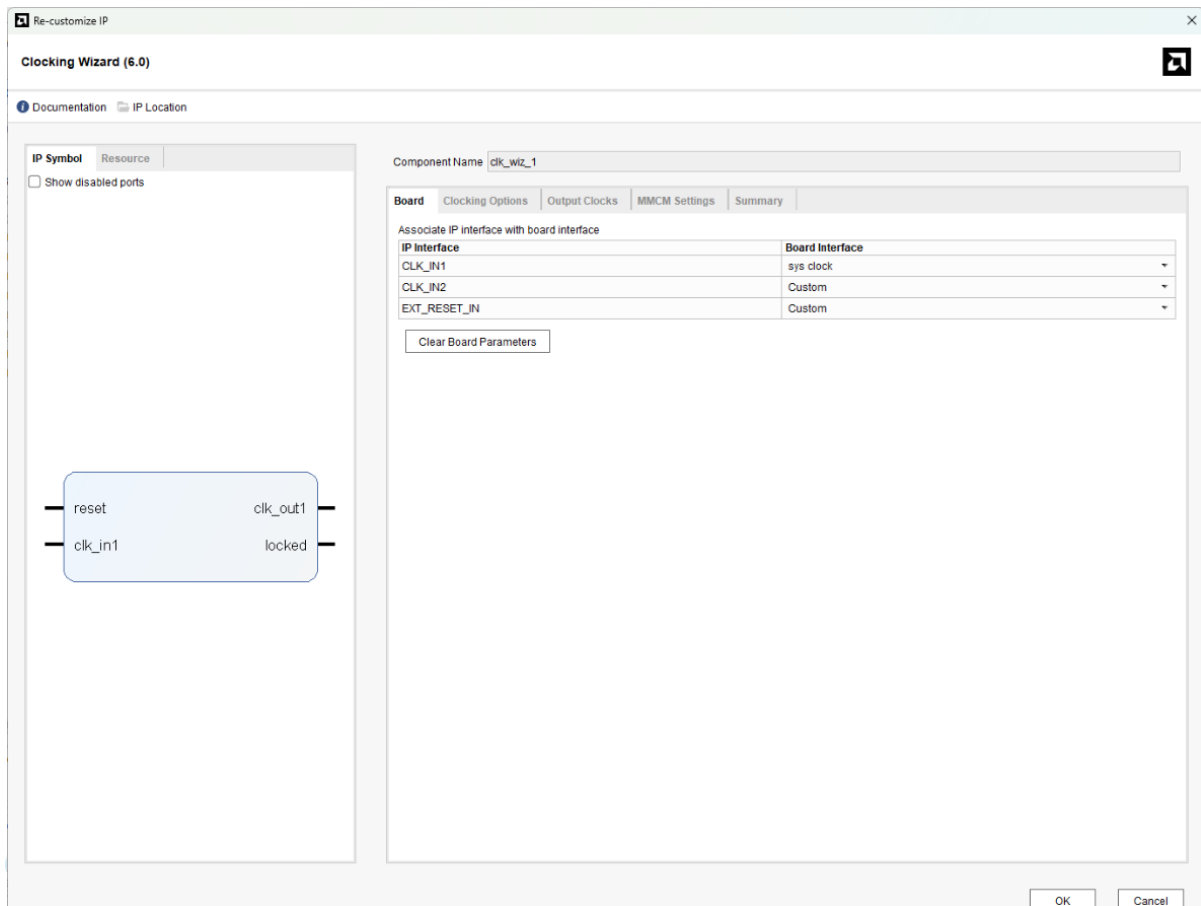


Click “**Run Block Automation**” present in the “**Designer Assistance available**” bar on the top left corner of the window to complete the design. Select the settings as shown in the following image. Click “**OK**” for Vivado to automatically configure the blocks for you.



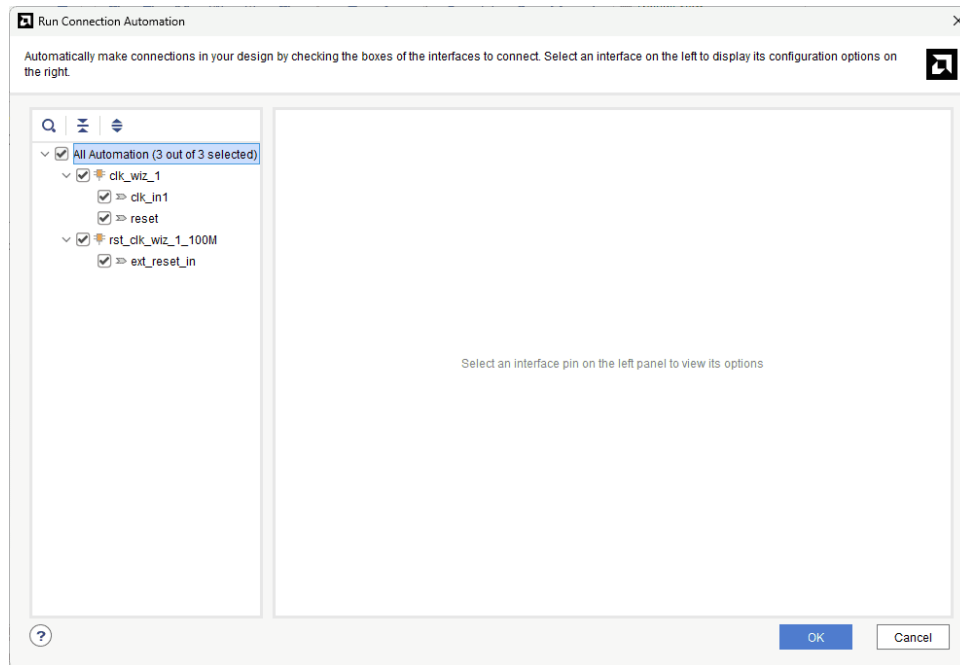
Step 4:

Double click “**Clocking Wizard**” IP and customize “**Board**” and “**Output Clocks**” settings as shown in the following image.

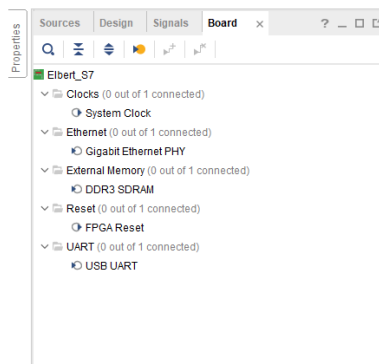


Step 5:

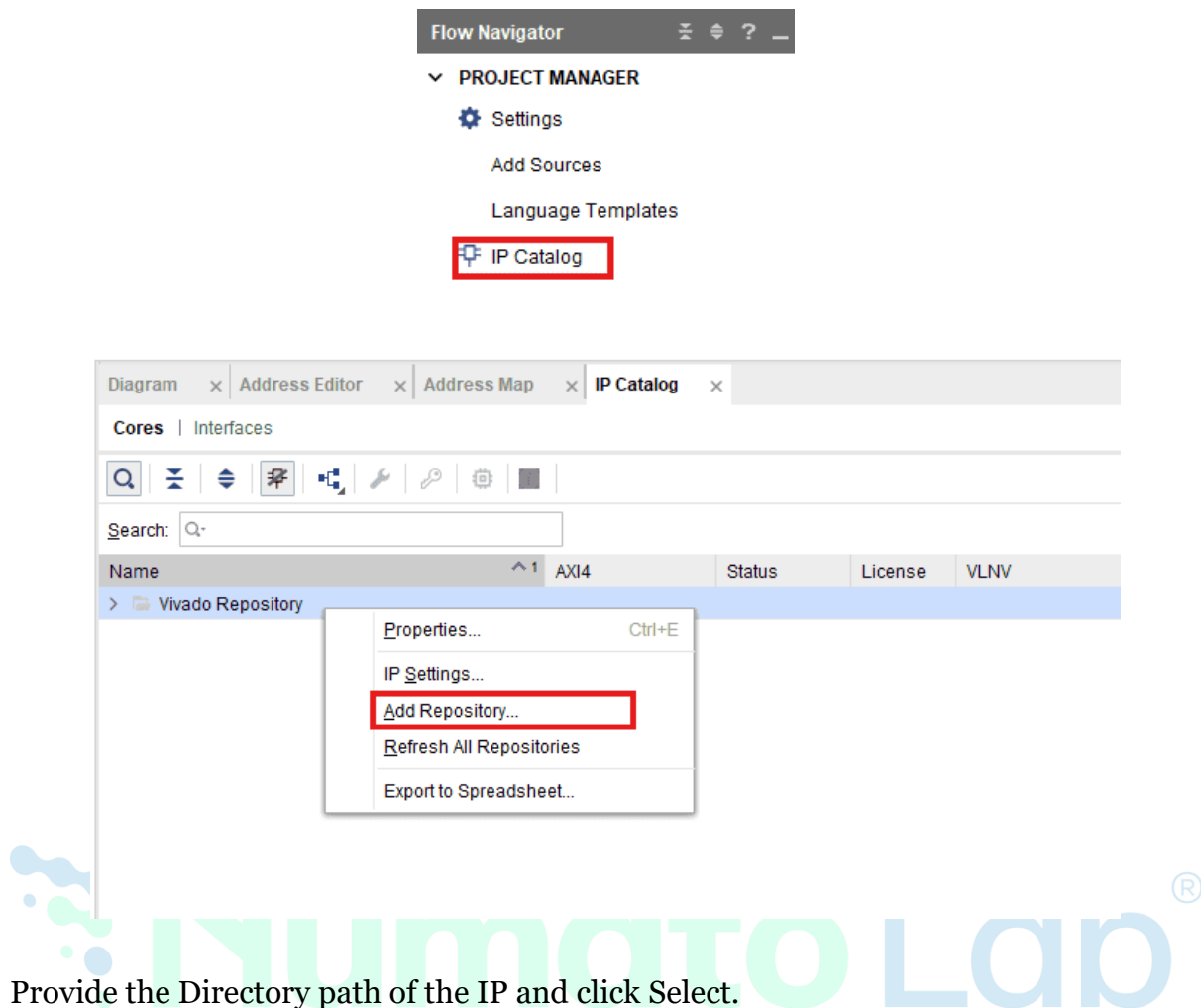
Run “**Connection Automation**” and select all the pins.

**Step 6:**

Go to the Board section, Drag and drop the USB UART from the Board section to the design.



Click on “Run Connection Automation” select all the pins and click ok.



Provide the Directory path of the IP and click Select.

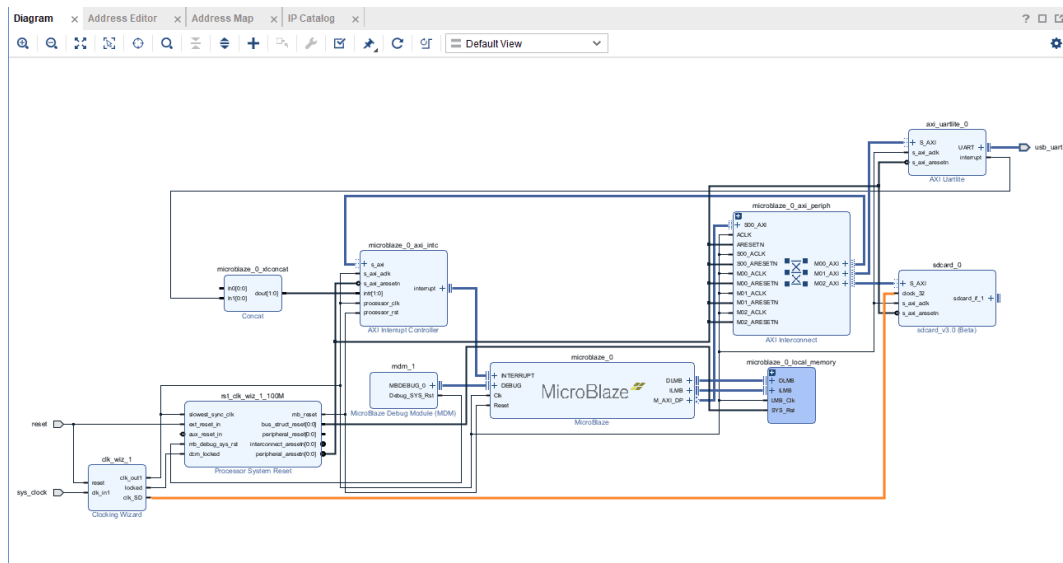
And Add the SDcard IP to the Block Design

Name	AXI4
User Repository (c:/projects/Elbert_S7/SD_card/sdcard)	
AXI Peripheral	
sdcard_v3.0	AXI4

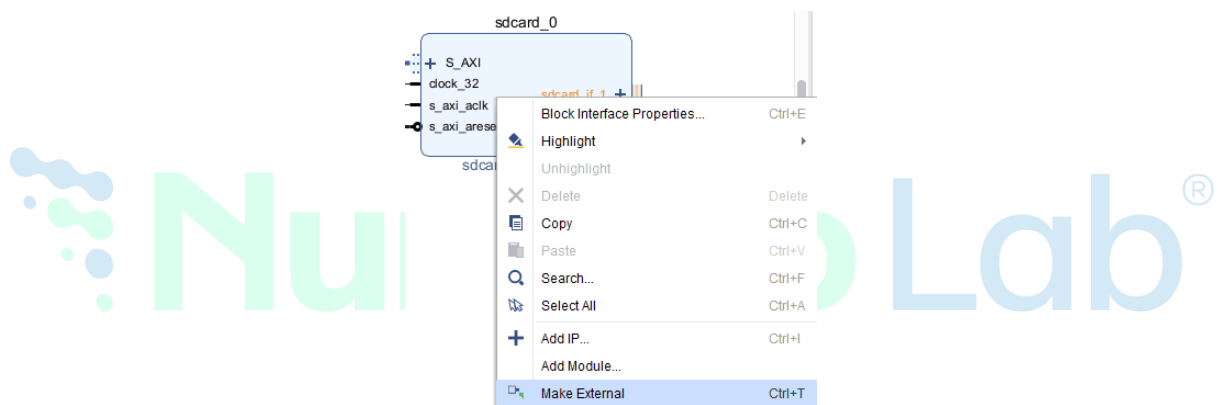
Step 8:

After adding the **SD card** Ip to the Block design, click on **Run Block Automation**.

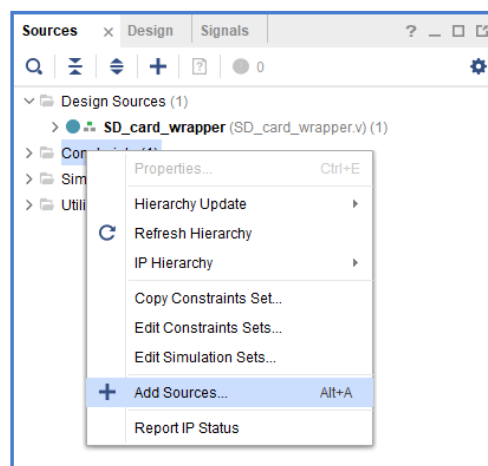
Now, connect the clock_32 pin of the SD card IP to the clk_SD pin of the Clocking wizard as shown in the below figure.



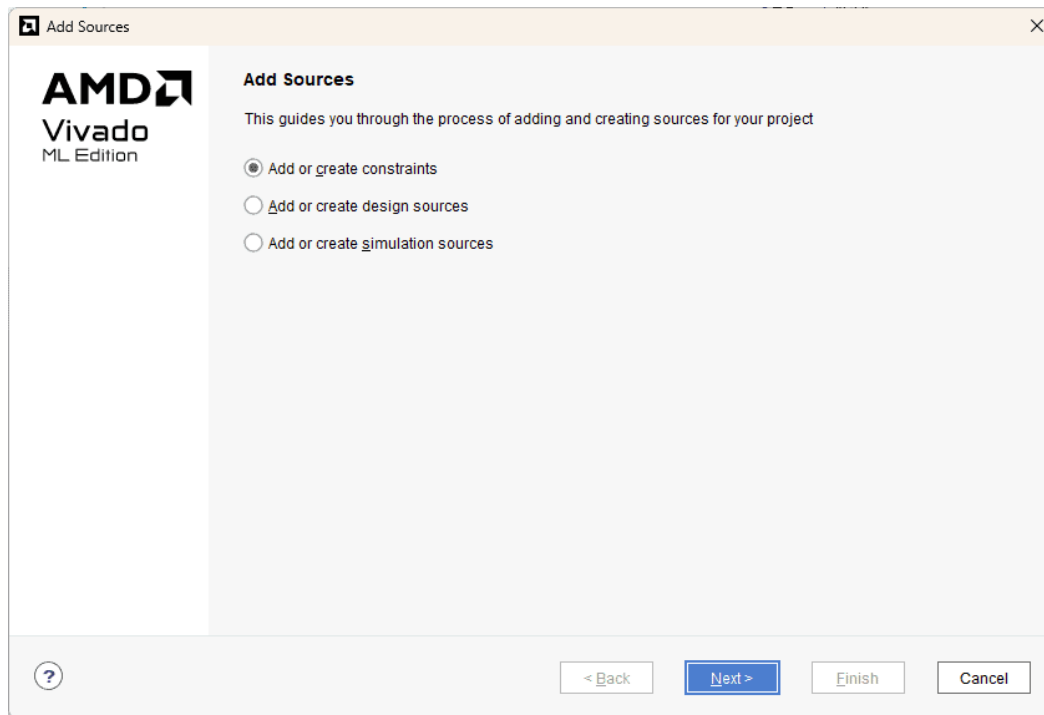
Step 9: Right click on Sdcard ip and click on Make External.



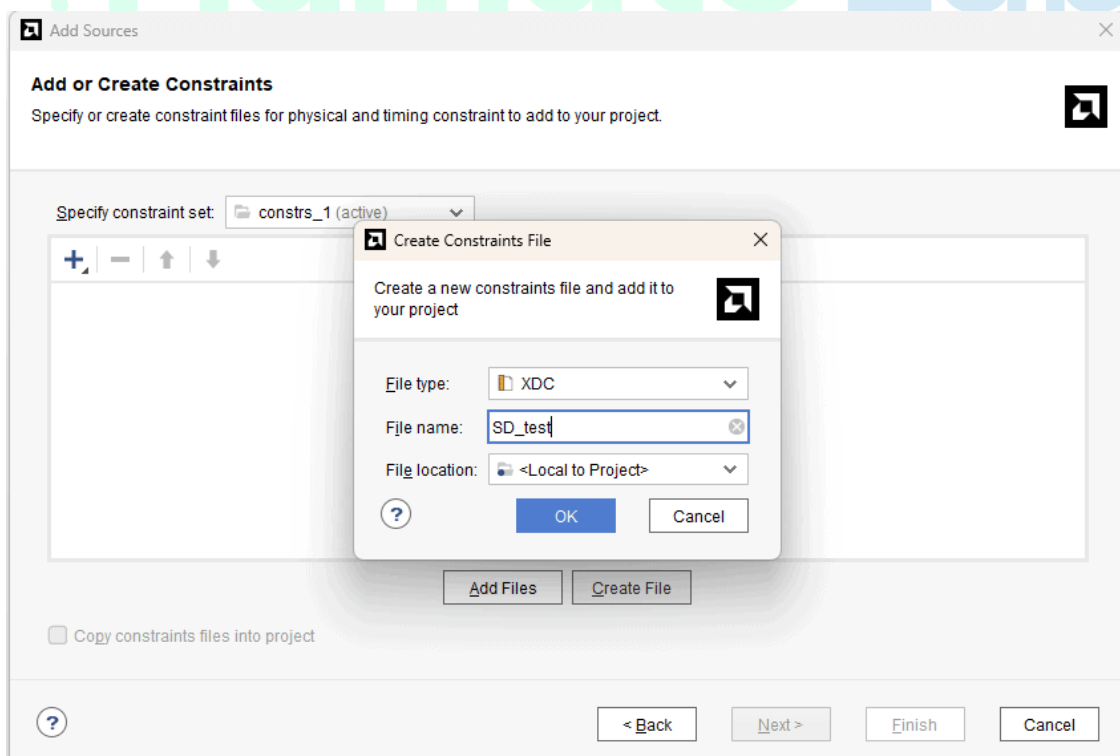
Step 10: In Sources tab of Vivado, Right-Click on ‘Constraints’ and click ‘Add Sources’.



Step 11: Once the “Add Sources” tab opens select “Add or create constraints” and click on “Next”.



Click on ‘Create File’ and give ‘SD_test’ as File name. Click ‘OK’ and ‘Finish’



Step 12:

Copy the following constraints in your constraints file and save it.

```
SET_PROPERTY -DICT {PACKAGE_PIN F14 IOSTANDARD LVCMOS33} [GET_PORTS
SDCARD_IF_1_0_SD_CLK]
```

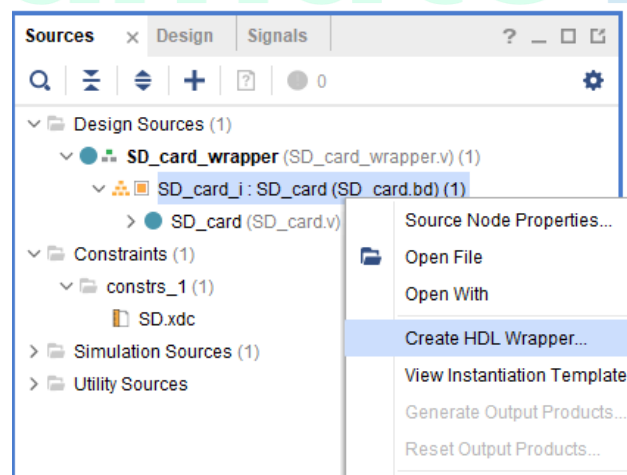
```
SET_PROPERTY -DICT {PACKAGE_PIN F15 IOSTANDARD LVCMOS33} [GET_PORTS
SDCARD_IF_1_0_SD_CS]
```

```
SET_PROPERTY -DICT {PACKAGE_PIN B17 IOSTANDARD LVCMOS33} [GET_PORTS
SDCARD_IF_1_0_SD_MISO]
```

```
SET_PROPERTY -DICT {PACKAGE_PIN A17 IOSTANDARD LVCMOS33} [GET_PORTS
SDCARD_IF_1_0_SD_MOSI]
```

Step 13:

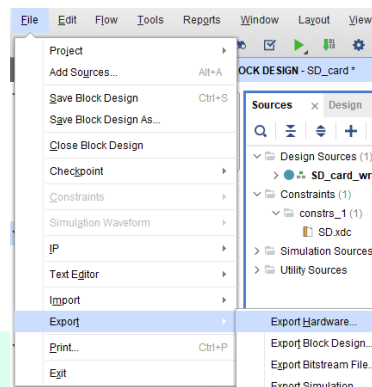
Right-click “SD_card” in the “Sources” window, and select “Create HDL Wrapper” from the popup menu. Click “OK” on the window that appears to finish generating a wrapper.

**Step 14:**

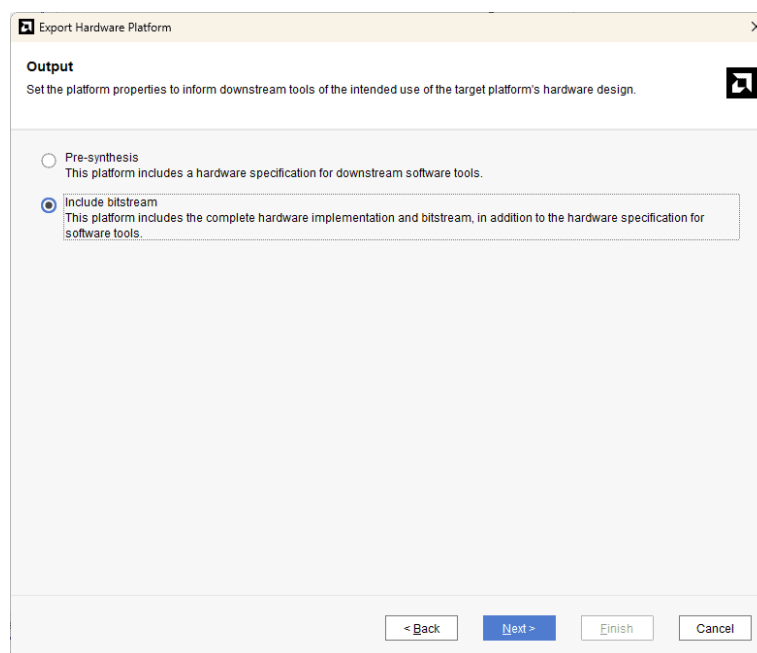
Click “Generate Bitstream” under the “Program and Debug” section to synthesize, implement and generate a bitstream.

- ▼ SYNTHESIS
 - ▶ Run Synthesis
 - > Open Synthesized Design
- ▼ IMPLEMENTATION
 - ▶ Run Implementation
 - > Open Implemented Design
- ▼ PROGRAM AND DEBUG
 - ▶ **Generate Bitstream**
 - > Open Hardware Manager

Step 15: After generating the bitstream successfully, select **Export -> Export Hardware** from the **File** menu. Click **Next**.



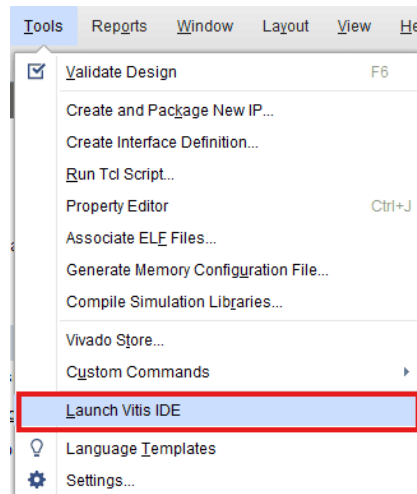
Select the “**include bitstream**” checkbox and click **Next**.



Provide the **XSA file name** and save it at a suitable **location**. Click **Next** and click **Finish** in the next dialog box.

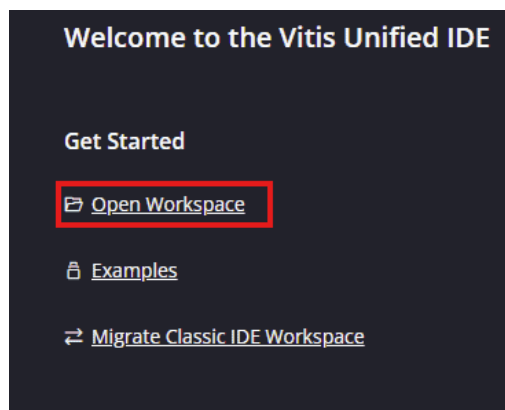
Step 16:

Select **Launch Vitis IDE** from the **Tools menu**.



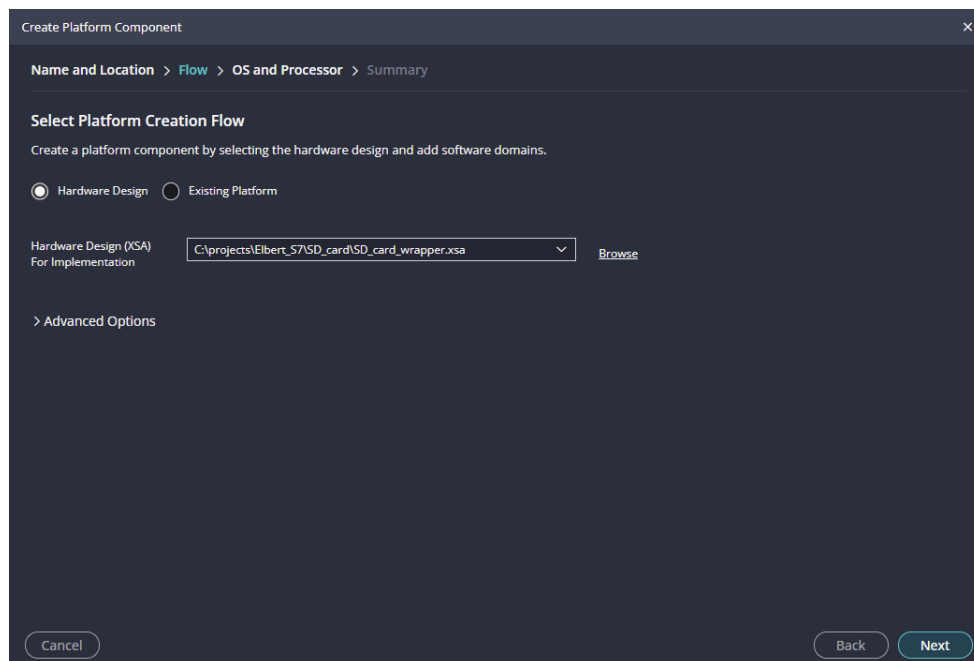
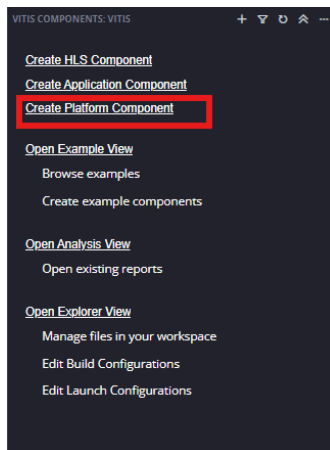
Step 17:

After Vitis Unified IDE window opens, click on “**Open Workspace**” and select necessary folder to keep the Vitis files.

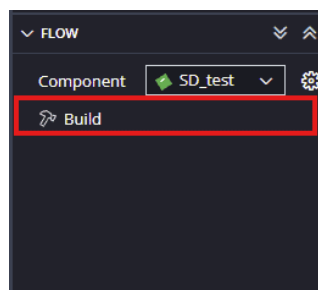


Step 18:

Create a new platform for the project, by selecting “**Create Platform Component**”, click “**Next**”, in the Flow tab select the XSA file saved using the step 20 and finally click “**Next**” and “**Finish**” respectively.

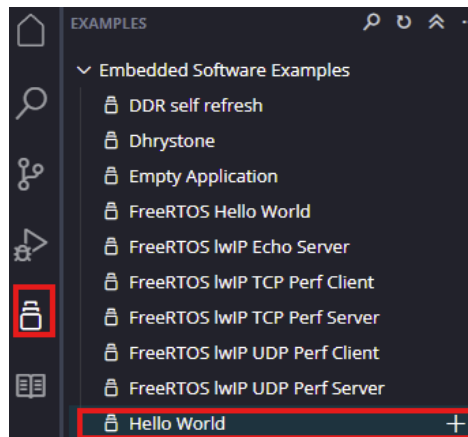


After successful creation of the platform, build the platform.

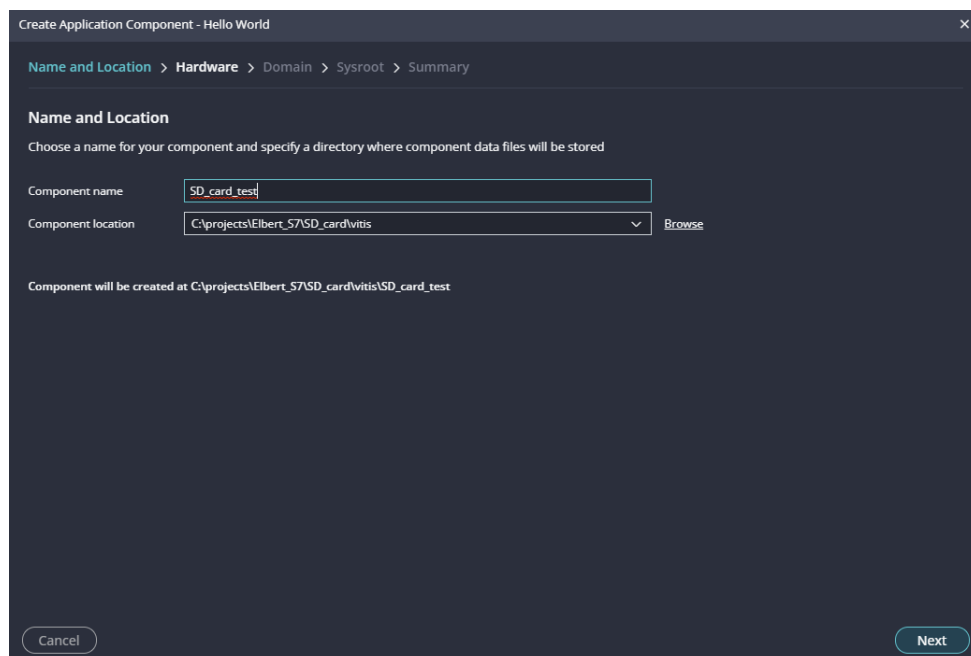


Step 19:

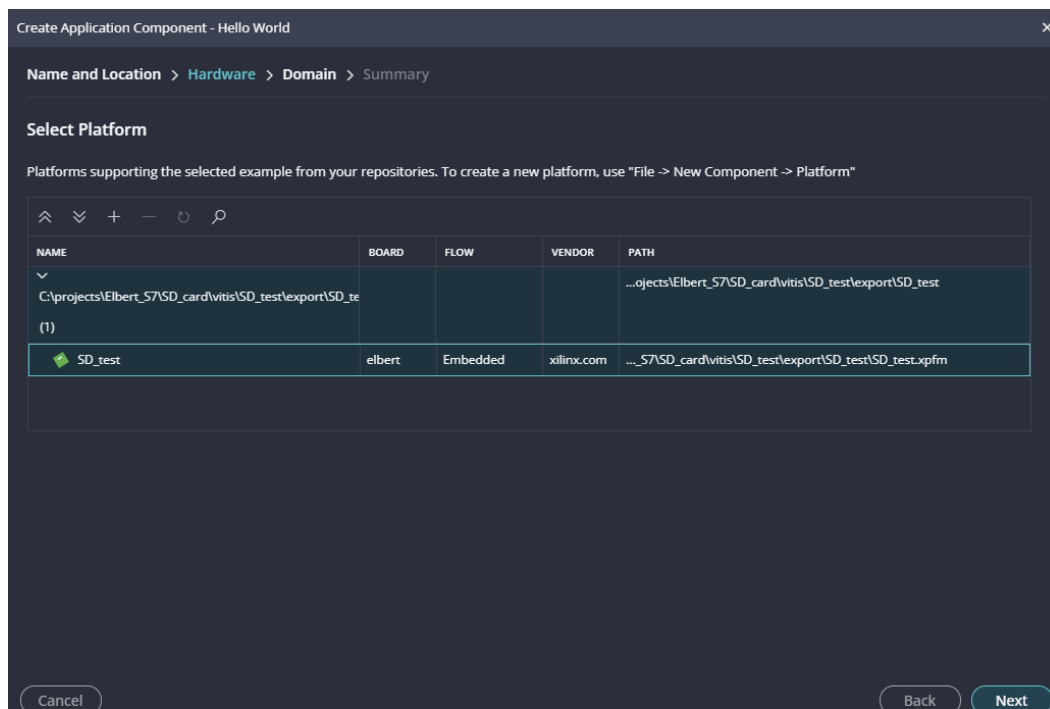
Next create the Helloworld Application component by selecting the “Helloworld” template from the “examples”,



In “Create Application Component” tab specify project name and location, click “Next”



Select newly created Platform and click “Next”.



Select the domain as “**Standalone_microblaze_o**” and click “**Next**” and click on “**Finish**”

Step 20: Download the SD_test.c file from [here](#), Copy the content of SD_test.c file to the helloworld.c file to test SD card .

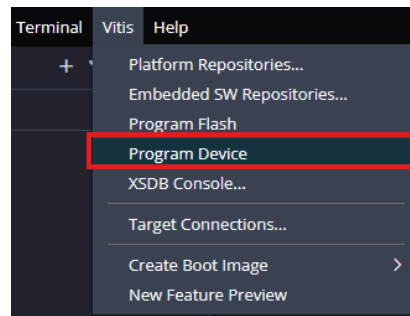
After adding the source file build the Project.

Step 21:

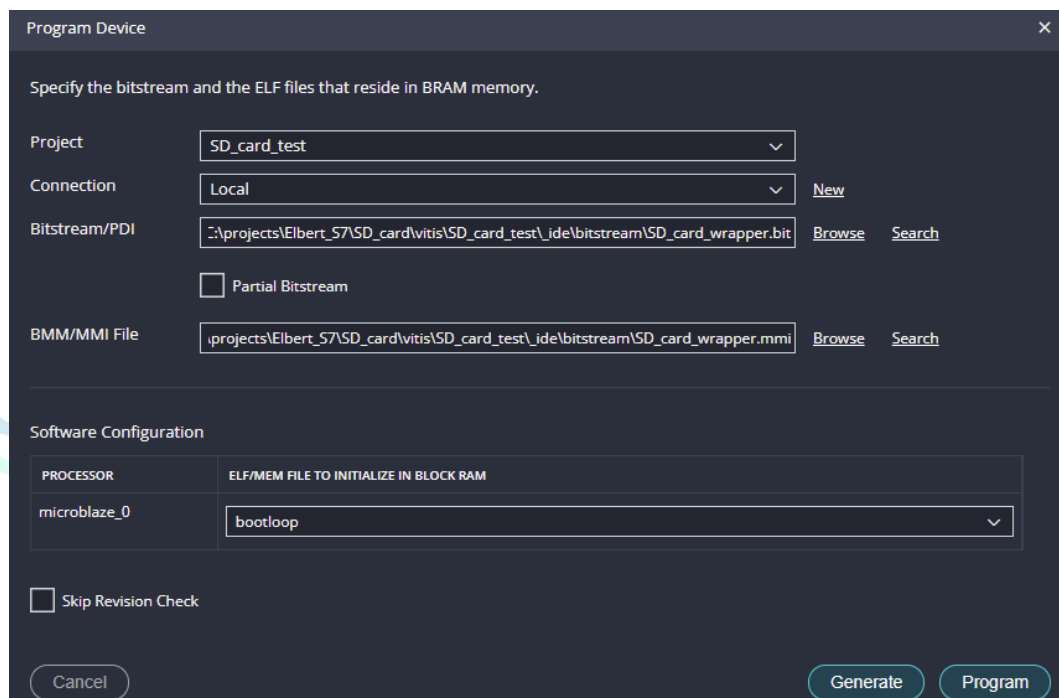
Once the build is completed successfully, power up Elbert S7 FPGA Development Board using an USB type C cable. and insert the SD card into the micro SD card slot of Elbert S7 FPGA Development Board.

Step 22:

Program the FPGA on Elbert S7 with a simple boot loop program by selecting the **Program Device** option from the **Vitis menu**.

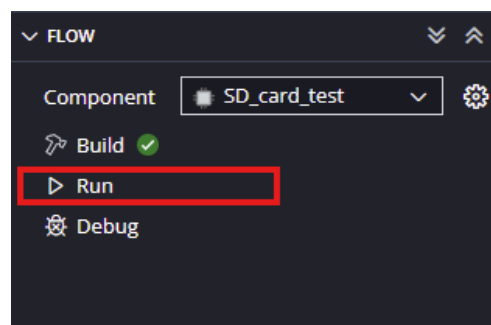


Once the “**Program Device**” window opens click on “**Program**”.



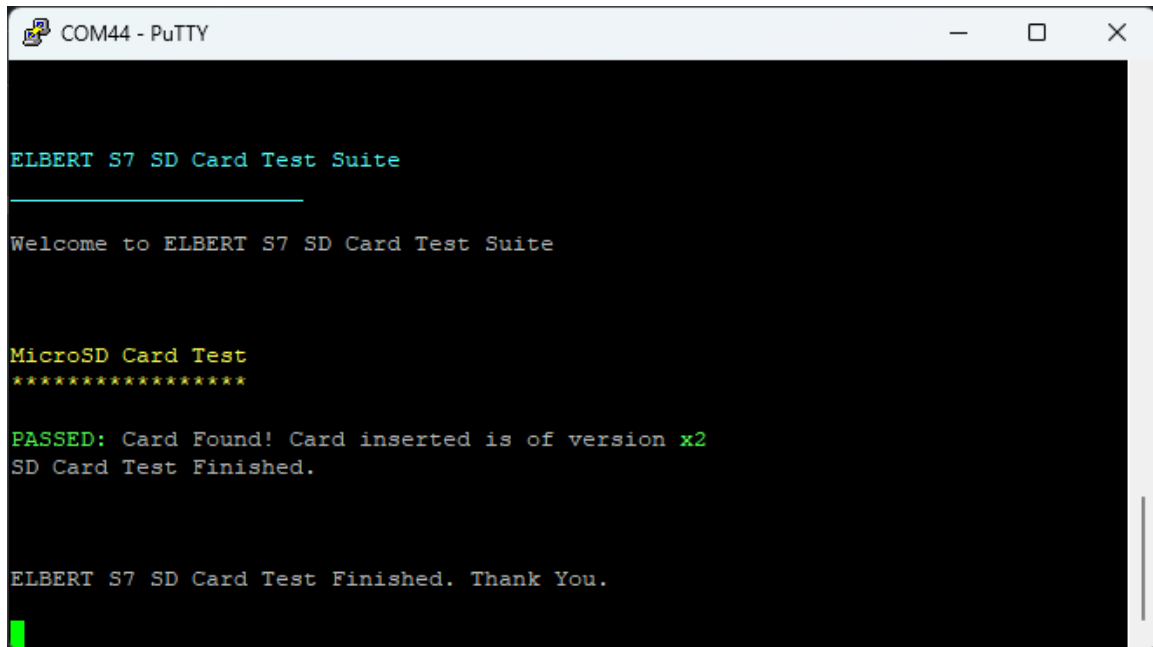
Step 23:

Meanwhile, open any serial terminal program (such as PuTTY, Teraterm etc) and open the port corresponding to Elbert S7 with a 9600 baud rate (the default baud rate given in UART IP). Program the board by selecting the “Run”.



Step 24:

If everything went well, Serial terminal would show the execution is Successful.



```
COM44 - PuTTY

ELBERT S7 SD Card Test Suite
_____
Welcome to ELBERT S7 SD Card Test Suite

MicroSD Card Test
*****

PASSED: Card Found! Card inserted is of version x2
SD Card Test Finished.

ELBERT S7 SD Card Test Finished. Thank You.
█
```



REFERENCES

➤ **PRODUCT LINKs:**

- [Product Page.](#)
- [User manual.](#)
- [Schematics.](#)
- [Xdc Constraints file.](#)

➤ **Tools Link:**

- [Vivado Design suite.](#)
- [PUTTY.](#)

