

## AUBoard-15P Development Kit Configuring the Clock Generator

Version 1.0



# Contents

1	Document Control .....	5
2	Version History .....	5
3	Introduction .....	6
3.1	Required Hardware and Software .....	7
4	Clock Configuration with Timing Commander .....	7
4.1	Configuring Timing Commander for the 8T49N241 .....	7
4.2	Modifying the Settings .....	8
4.3	Export the New Settings for the EEPROM .....	10
5	EEPROM Programming with Vivado and Vitis .....	11
5.1	Building the Vivado project .....	12
5.2	Creating the Vitis project .....	13
5.3	Run on Hardware .....	17
6	Vivado Design for Clock Verification and Usage .....	19
6.1	Creating the Vivado design .....	19
6.2	Running the Reference Design .....	25
7	Conclusion .....	26
8	Getting Help and Support .....	27

## Figures

Figure 1 – Clock Generator (U57) and EEPROM (U58).....	6
Figure 2 – Timing Commander: Open Settings File .....	8
Figure 3 – Timing Commander: Save As .....	9
Figure 4 – Timing Commander: USRCLK Modification .....	10
Figure 5 – Timing Commander: EEPROM Editor .....	11
Figure 6 – Vivado Design: Block Design .....	13
Figure 7 – Vitis: Select Platform .....	14
Figure 8 – Vitis: Import Files.....	15
Figure 9 – Updated at24aa025_config.h File .....	16
Figure 10 – Vitis: Successful Build .....	17
Figure 11 – Board Setup .....	17
Figure 12 –Run Application on Hardware.....	18
Figure 13 – Application Output.....	18
Figure 14 – Programming Done .....	19
Figure 15 – Vivado: Set Repo Path.....	20
Figure 16 – Vivado: Select Board Definition.....	21
Figure 17 – Vivado: Clocking Wizard Configuration .....	22
Figure 18 – Vivado: Block Design .....	23
Figure 19 – Vivado: HDL Wrapper .....	23
Figure 20 – Vivado: XDC Constraints.....	23
Figure 21 – Vivado: Bitstream Generation Failure .....	24
Figure 22 – UltraScale+ User Guide: HDIO Global Clocks .....	24
Figure 23 – Updated XDC Constraint.....	25
Figure 24 – Board: D32 LED Blinking.....	26

## 1 Document Control

**Document Version:** 1.0

**Document Date:** 25 January 2025

## 2 Version History

Version	Date	Comment
1.0	01/25/2025	Initial Release

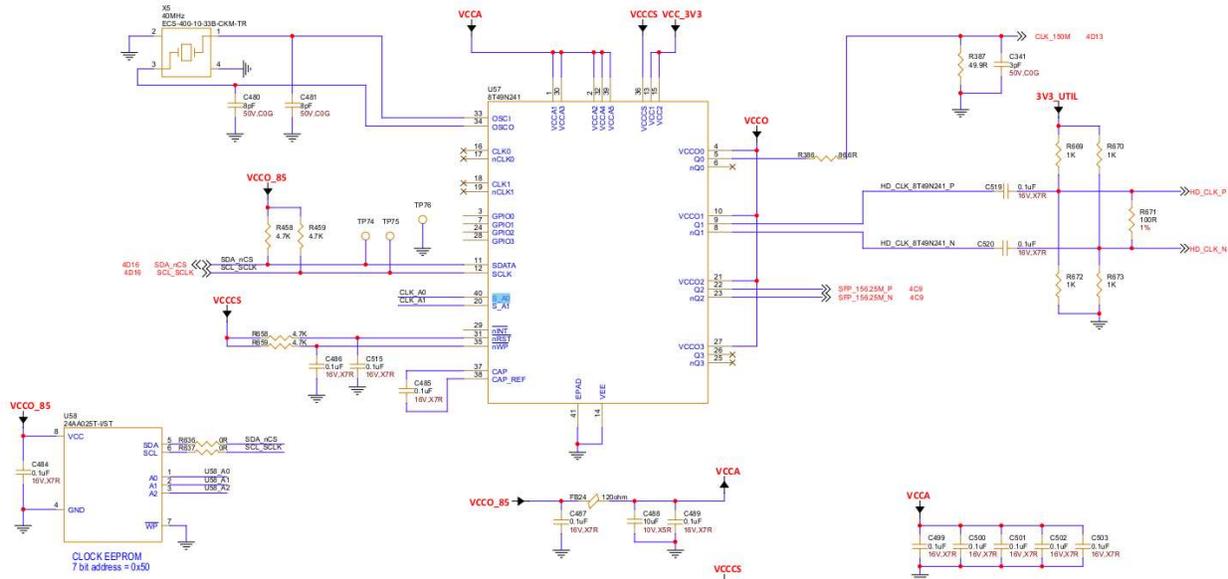
### 3 Introduction

The AUBoard-15P FPGA Development Kit features two programmable clock sources that provide flexible clock generation for various applications. The first clock source, device **U56**, is used to forward a jitter attenuated version of a differential clock generated by the HDMI FPGA design. The second clock source, device **U57**, provides several clock outputs, including a GTH transceiver reference clock, a general-purpose programmable LVDS clock, and the FPGA EMCCLK used for external configuration.

For this guide, we will focus specifically on configuring the second programmable clock source, **U57**, through its associated EEPROM (**24AA025T**). This EEPROM stores the configuration data that customizes the clock generator settings, which are then applied during power-up.

The **8T49N241** clock generator is a programmable device that supports both single-ended and differential outputs. The configuration values for the EEPROM will be generated using the Timing Commander Tool from Renesas,

A schematic of the AUBoard-15P board, highlighting the **U57** and **U58** components, is shown in Figure 1 below.



**Figure 1 – Clock Generator (U57) and EEPROM (U58)**

This guide is divided into three main sections to help you configure and verify the clock settings:

- **Clock Configuration with Timing Commander:** Instructions on configuring the clock settings and generating the EEPROM content using Timing Commander.
- **EEPROM Programming with Vivado and Vitis:** Detailed steps for rebuilding the Vivado design, re-creating a standalone C application, and programming the EEPROM on the AUBoard-15P board.
- **User Clock Verification and Usage:** Instructions for creating a Vivado design to verify that the “general-purpose programmable LVDS clock” also referred to as User clock has been programmed correctly, and to demonstrate how to use the clock in your application.

By following the steps in this guide, you will be able to efficiently configure and verify the clock settings, enabling reliable clock generation on your AUBoard-15P applications.

### 3.1 Required Hardware and Software

To follow this guide and successfully configure and verify the clock settings on the AUBoard-15P ensure you have the necessary tools and components:

- The AUBoard-15P FPGA Development Kit
- A Micro-B USB Debug Cable for USB-UART / JTAG
- Vivado Design Suite and Vitis Unified Software Platform installed on your computer (2024.1 version used in this guide). Go to <https://www.xilinx.com/support/download.html> for download links and help.
- Renesas Timing Commander Tool installed on your computer (1.18 version used in this guide). Go to <https://www.renesas.com/en/software-tool/timing-commander> for download links and help.
- The source utilized this guide – **AUBoard-15P-clock-configuration.zip** - containing the necessary files to rebuild the examples. This archive is available on the reference design tab on the AUBoard-15P website located at the following short URL: <http://avnet.me/AUBoard-15P>

## 4 Clock Configuration with Timing Commander

This section focuses on using the Renesas Timing Commander Tool to configure the programmable clock generator (U57) on the AUBoard-15P. The Timing Commander Tool simplifies the process of generating the necessary EEPROM content for the 8T49N241 clock generator, enabling precise control of its output frequencies.

We will guide you through the steps to define the clock settings, validate the configuration, and export the resulting EEPROM data file. This file will later be used to program the 24AA025T EEPROM connected to U57, ensuring that the desired clock frequencies are applied at power-up.

By the end of this section, you will have a complete configuration file ready for programming, tailored to your application requirements.

### 4.1 Configuring Timing Commander for the 8T49N241

To set up Timing Commander for the 8T49N241 clock generator, follow these steps:

- **LAUNCH TIMING COMMANDER** and click on “**OPEN SETTINGS FILE**” from the main menu.
- For the **SETTING FILE**, select the example provided in the source archive:
  - `AUBoard-15P-clock-configuration\timing_commander\IDT8T49N241_20241205_011247_U57_300MHz.tcs`
- Next, click on the **FOLDER ICON** labelled “**BROWSE FOR A PERSONALITY THAT SUPPORTS THE SETTING FILE**”.
- Locate and select the corresponding personality file from the source archive:

- **AUBoard-15P-clock-configuration\timing\_commander\8T49N24x\_V1.7.3.tcp**
- Once the settings file and personality file are loaded, the Timing Commander interface should appear as the example shown in the figure below:



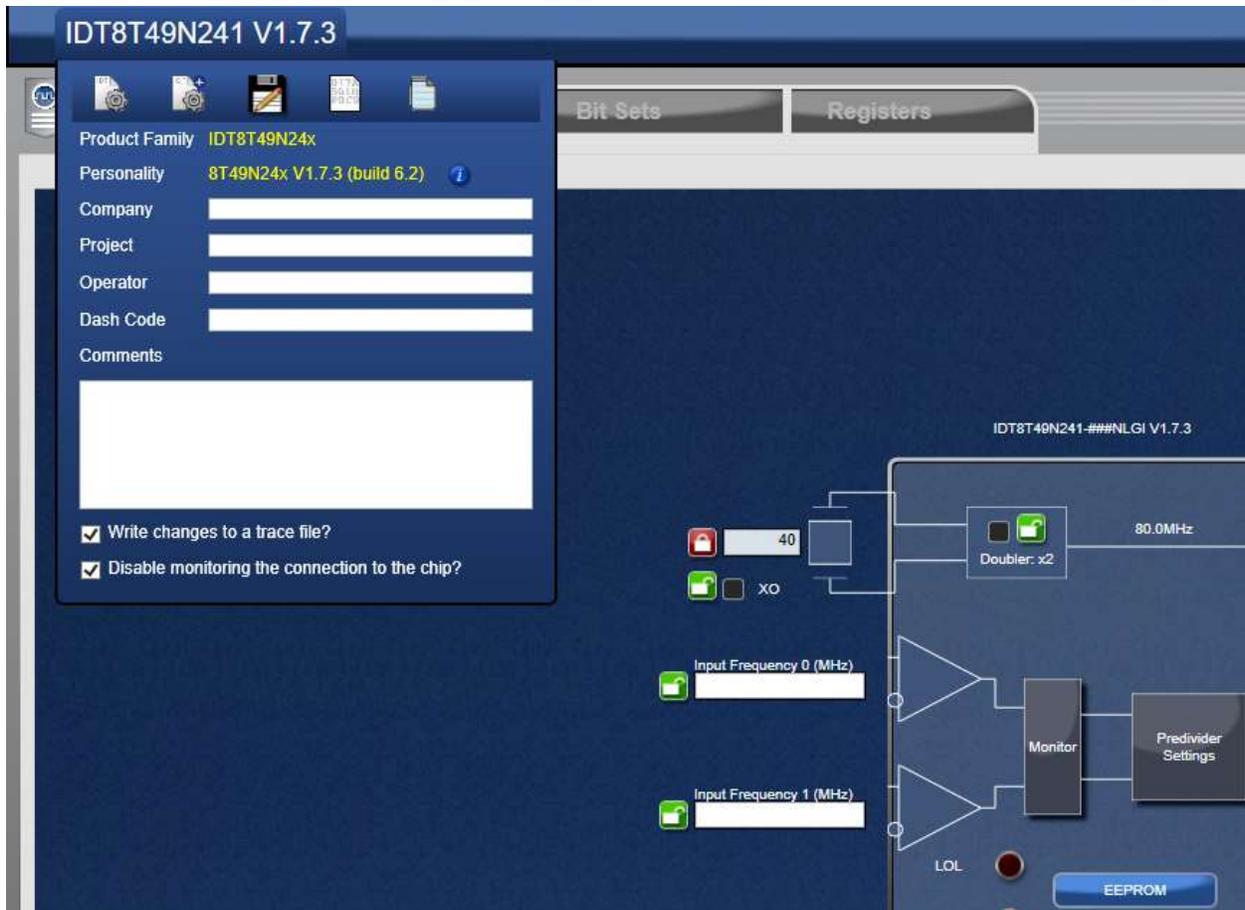
**Figure 2 – Timing Commander: Open Settings File**

- Proceed with this setup by clicking on the **“OPEN”** button

## 4.2 Modifying the Settings

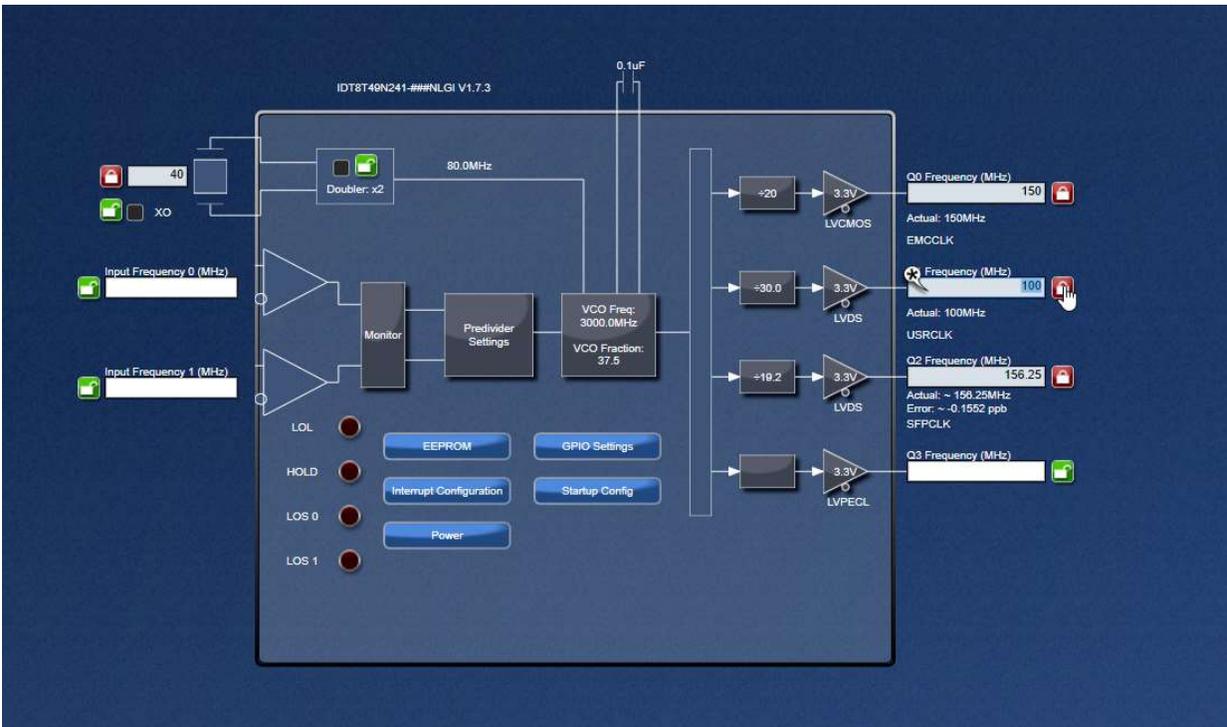
To customize the example configuration and set the user clock frequency to **100 MHz**, follow these steps:

1. Save the Current Settings File:
  - a. Click on the **“IDT8T49N241 V1.7.3”** text at the top left corner of the Timing Commander window.
  - b. Click on the **FLOPPY DISK ICON** to save the file with a new name (e.g., IDT8T49N241\_my\_settings):
  - c. Click **“YES”** when asked to include the register map.



**Figure 3 – Timing Commander: Save As**

2. Unlock the User Clock Frequency Setting:
  - a. Locate the **RED LOCK ICON** associated with the user clock frequency parameter: “**Q1 FREQUENCY**” for **USRCLK**.
  - b. Click on the lock icon to enable editing of this setting.
3. Enter **100** in the input field for the user clock frequency.
4. Click on the **LOCK ICON** again to lock the clock setting and prevent further changes. The Timing Commander interface should appear as the example shown below:



**Figure 4 – Timing Commander: USRCLK Modification**

### 4.3 Export the New Settings for the EEPROM

Once you've modified the user clock frequency, it's time to export the updated settings for the EEPROM.

**IMPORTANT:** Timing Commander may not always register the modifications immediately. A quick way to verify that the changes are effective is by checking the **CRC8 VALUE**.

#### 1. Verify the Modifications with CRC8:

- a. Click on the EEPROM button in the top right corner of the interface.
- b. Select **EDIT** or **OVERRIDE** to view the EEPROM configuration dump.

#### 2. Check the CRC8 Value:

- a. At the bottom of the EEPROM Editor window, you will see the **CRC8 VALUE**.
- b. The CRC8 value should be **DIFFERENT** from the example settings you started with, which had a CRC8 of **0x79**.
- c. After modifying the Q1 frequency to **100 MHz**, the CRC8 value should now be **0xD9**, as shown in the image below:

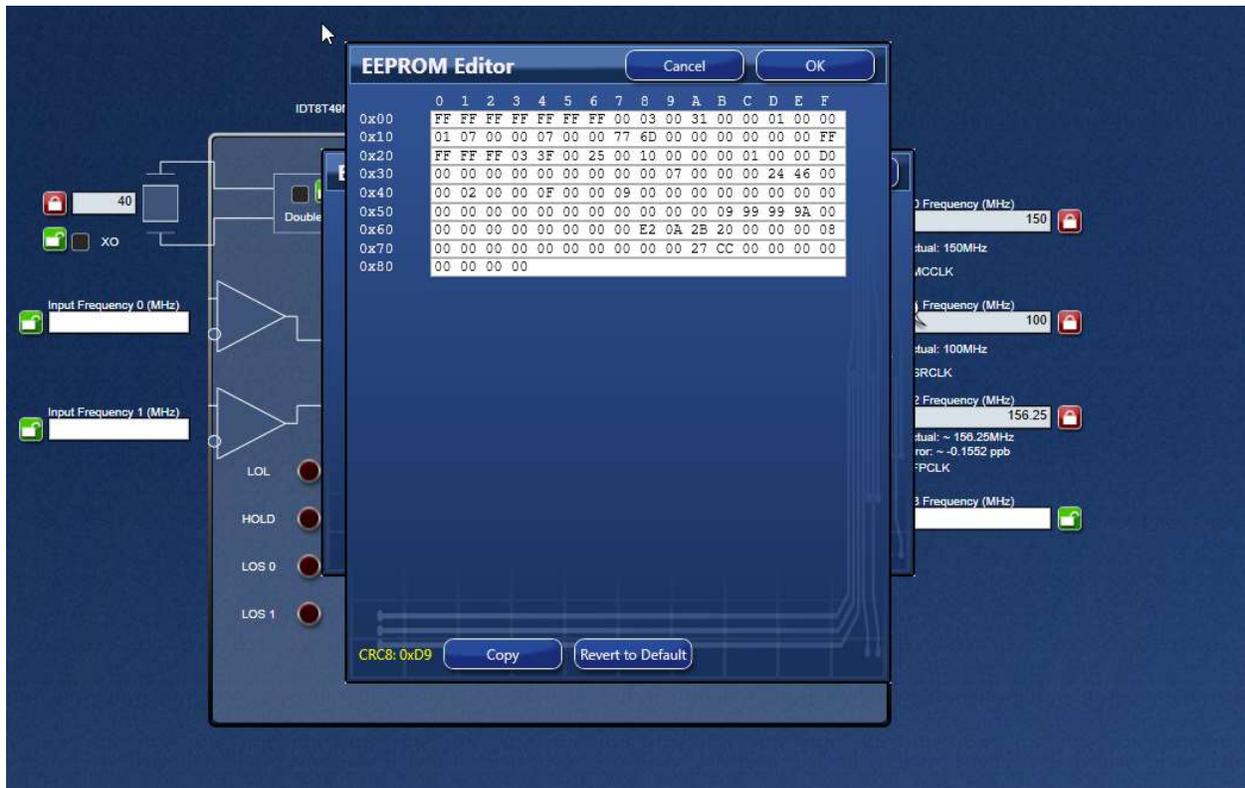


Figure 5 – Timing Commander: EEPROM Editor

3. Export the Settings:

- a. If the **CRC8 value is correct**, click **OK** and then click **Export to File** to save the settings. You can keep the default file name: **IDT8T49N241\_my\_settings\_EEPROM**.
- b. If the **CRC8 value is incorrect** (either unchanged or modified when clicking on the **Revert to Default** button), click **Cancel**, then restart the verification process by clicking the **“EDIT”** or **“OVERRIDE”** button, or try modifying the Q1 frequency again.

4. Verify the Generated File:

- a. Open the generated file **IDT8T49N241\_my\_settings\_EEPROM.txt**.
- b. You should see a sequence of hexadecimal values, with the last one being the **CRC8 value** (which should be **0xD9** in our case).

## 5 EEPROM Programming with Vivado and Vitis

In this section, we will create a hardware design in Vivado featuring a MicroBlaze processor, a UART for communication, and IIC to interface with the EEPROM. This design will provide the necessary infrastructure to program the EEPROM on the AUBoard-15P Development Kit.

Using Vitis, we will develop a standalone application to program the EEPROM with the configuration file generated in the previous chapter. This process ensures that the user clock is programmed accurately and is ready for verification.

## 5.1 Building the Vivado project

**NOTE:** Much of reference designs created by Tria are created/scripted for use on Linux workstations. It is possible to perform these tasks on Windows machines, but for this reference design example the guide shows implementation on a Linux machine.

On a Linux computer with Vivado version 2024.1 installed:

1. Open a Linux terminal.
2. **Set the Environment Variable**

Set the `GUIDE_HOME` variable to the location of the downloaded project files:

```
$ export GUIDE_HOME=</path/to/downloaded/zipfile>/AUBoard-15P-  
clock-configuration
```

3. Change directory to **\$GUIDE\_HOME/**. Clone the board definition file repository:

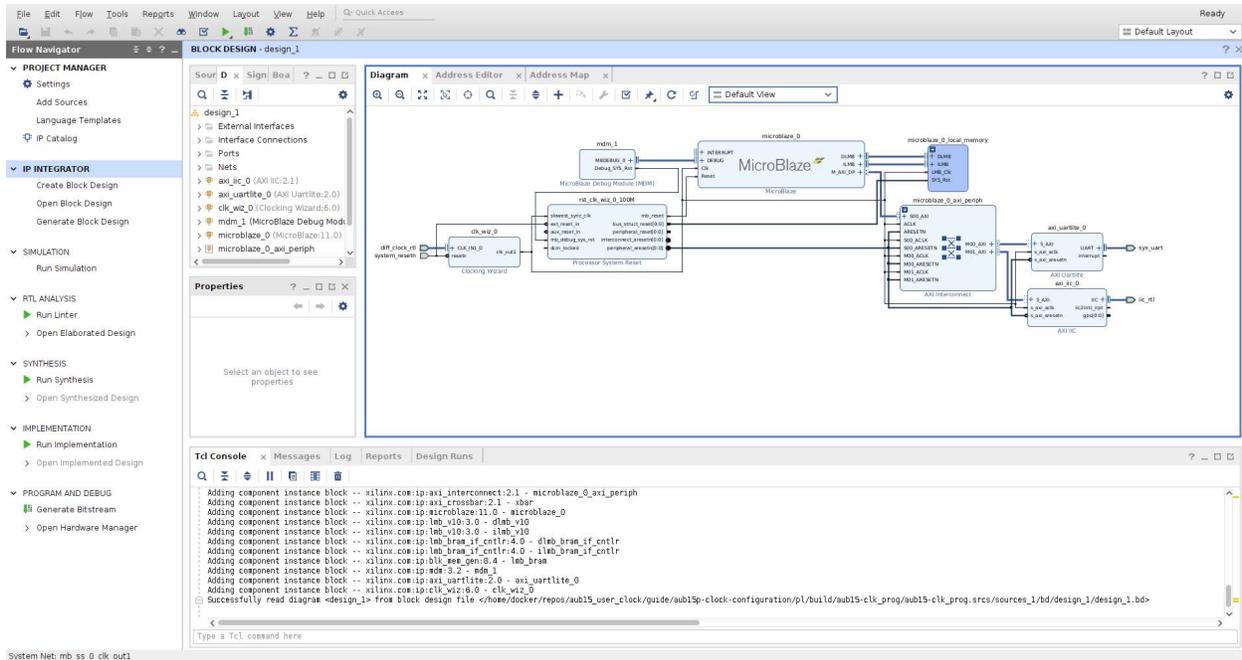
```
$ git clone https://github.com/Avnet/bdf.git
```

4. Change directory to **\$GUIDE\_HOME/pl**, run:

```
$ vivado -source build.tcl -tclargs --repo-paths ../bdf
```

This command should open Vivado and create the design in the **\$GUIDE\_HOME/pl/build** folder. The design includes a Microblaze, an AXI Uartlite for serial communication and an AXI IIC for accessing the EEPROM.

If you click on “**OPEN BLOCK DESIGN**” (Flow Navigator panel), you should see the following:



**Figure 6 – Vivado Design: Block Design**

5. Click on “**GENERATE BITSTREAM**” (Flow Navigator panel)
6. Exporting the resulting generate hardware design file:
  - a. Once the bitstream is generated, click on **FILE -> EXPORT -> EXPORT HARDWARE...**, then click on **NEXT**.
  - b. Check the “**INCLUDE BITSTREAM**” option and click on **NEXT**.
  - c. You can then change the name and location of the generated XSA. By default, the XSA will be written in **\$GUIDE\_HOME/pl/build/aub15-clk\_prog/design\_1\_wrapper.xsa**. Click on **NEXT** and then **FINISH**.
  - d. Once the XSA is generated, you can close Vivado.

## 5.2 Creating the Vitis project

To create the Vitis project based on the XSA previously generated:

1. Change directory to **\$GUIDE\_HOME/vitis** and create a **WORKSPACE** directory:

```
$ cd $GUIDE_HOME/vitis

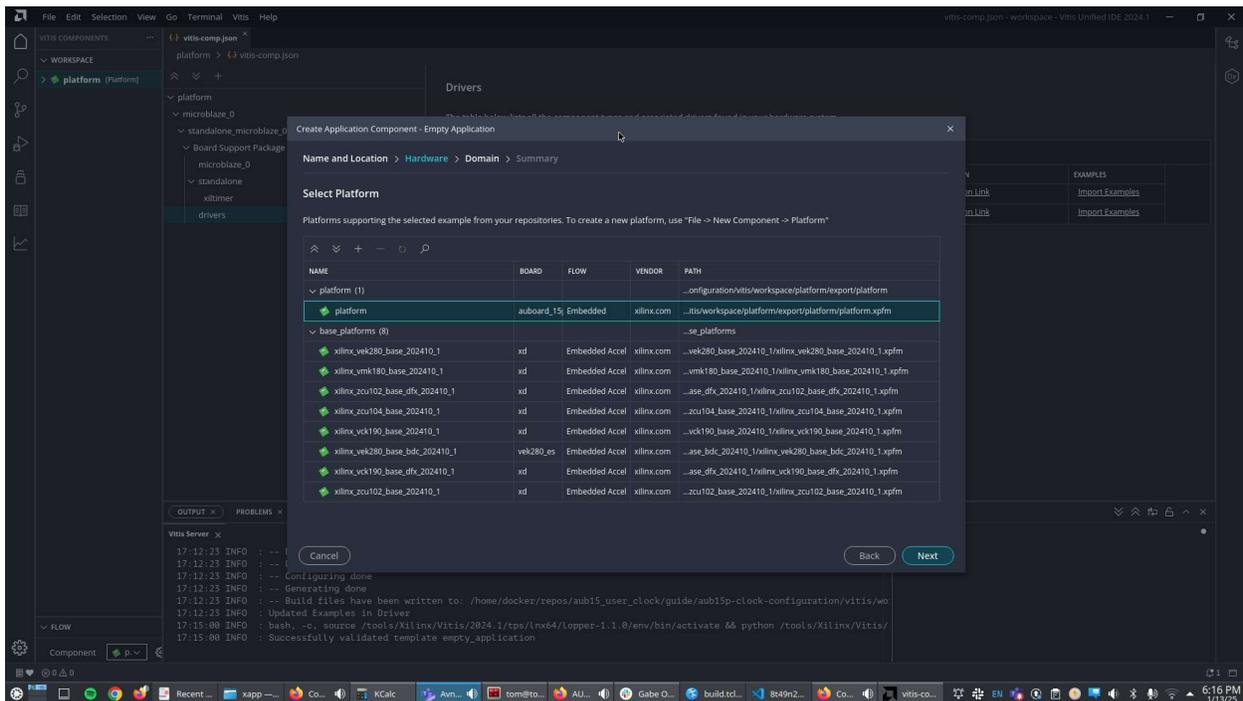
$ mkdir workspace

$ cd workspace
```

- Open Vitis with:

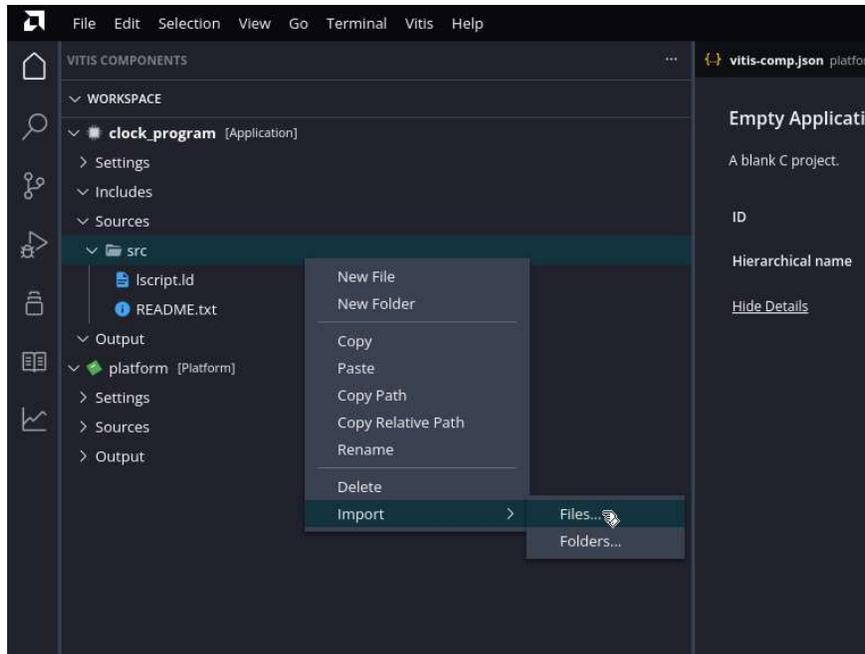
```
$ vitis &
```

- On the left panel, click on “**OPEN WORKSPACE**”, and select the **\$GUIDE\_HOME/vitis/workspace** folder.
- On the left panel, click on “**CREATE PLATFORM COMPONENT**”, then select **NEXT**.
- Click on **BROWSE** to select an XSA. Select the XSA created in the previous section (named by default): **\$GUIDE\_HOME/pl/build/aub15-clk\_prog/design\_1\_wrapper.xsa**
- Leave the Operating system as **STANDALONE**, and click on **NEXT**, and **FINISH**.
- Click on **FILE -> NEW COMPONENT -> APPLICATION**.
- Write “**clock\_program**” as the **COMPONENT NAME** and click on **NEXT**.
- Select the **PLATFORM** that was created in the previous steps:



**Figure 7 – Vitis: Select Platform**

- Click on **NEXT**. On the Select Domain page, click on the **standalone\_microblaze\_0** domain, then **NEXT**, and **FINISH**.
- On the left panel, under the new “**clock\_program**” Application, Sources. Right click on **SRC**, and click on **IMPORT -> FILES...**



**Figure 8 – Vitis: Import Files**

12. **SELECT ALL** the files from the **\$GUIDE\_HOME/vitis/src/** folder.
13. On the left panel, click on the **at24aa025\_config.h** file from the **SRC** directory to modify it:
  - a. Replace the content of the **AT24AA025\_IDT8T49N241\_CFG** array with the data found in the **IDT8T49N241\_my\_settings\_EEPROM.txt** file generated by Timing Commander (Section 4.3 of this document).
  - b. The **at24aa025\_config.h** file should now look like the following figure:

```

#ifndef __AT24AA025_CONFIG_H_
#define __AT24AA025_CONFIG_H_

#define AT24AA025_CONFIG_SIZE 0x85

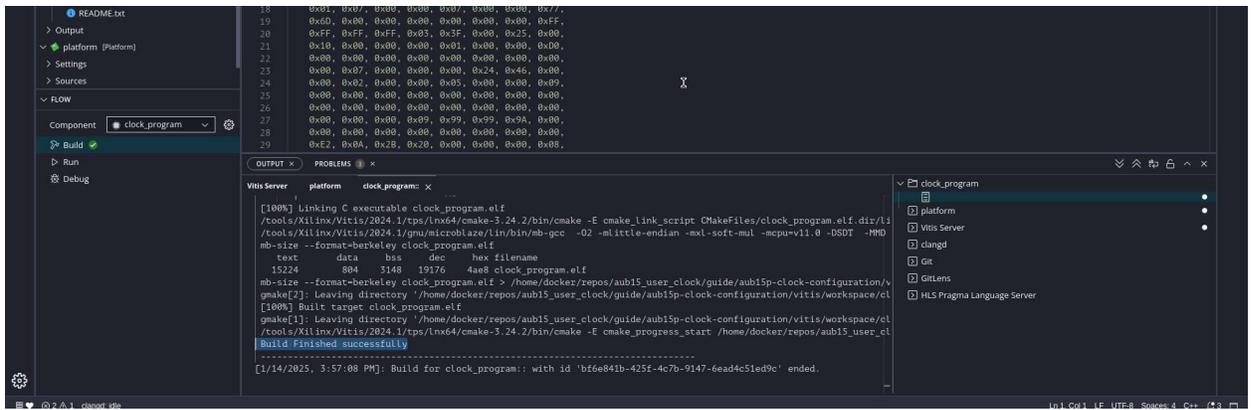
u8 AT24AA025_IDT8T49N241_CFG[AT24AA025_CONFIG_SIZE] = {
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x03, 0x00, 0x31, 0x00, 0x00, 0x01, 0x00, 0x00,
0x01, 0x07, 0x00, 0x00, 0x07, 0x00, 0x00, 0x77,
0x6D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
0xFF, 0xFF, 0xFF, 0x03, 0x3F, 0x00, 0x25, 0x00,
0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0xD0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x07, 0x00, 0x00, 0x00, 0x24, 0x46, 0x00,
0x00, 0x02, 0x00, 0x00, 0x0F, 0x00, 0x00, 0x09,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x09, 0x99, 0x99, 0x9A, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xE2, 0x0A, 0x2B, 0x20, 0x00, 0x00, 0x00, 0x08,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x27, 0xCC, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xD9
};

#endif

```

**Figure 9 – Updated at24aa025\_config.h File**

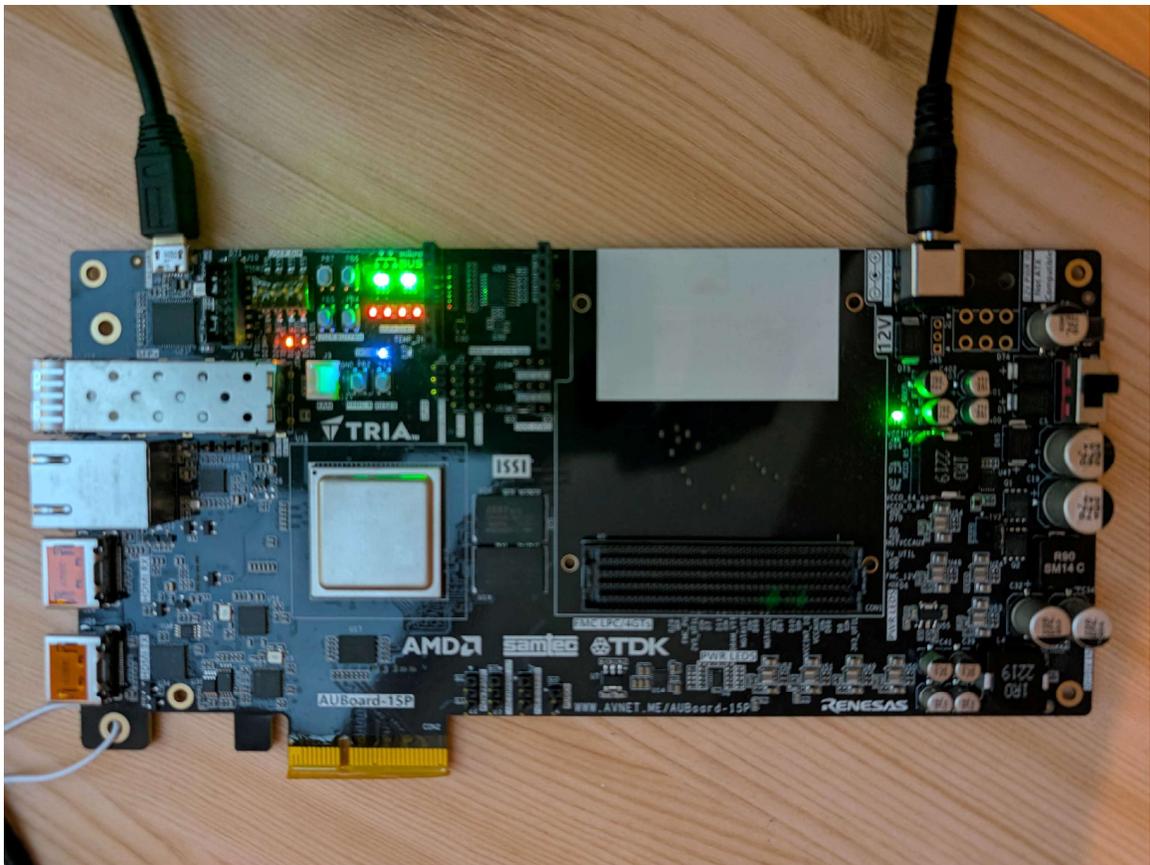
- On the left panel, in the Flow section, click on **BUILD**. The build should be successful, as shown in the figure below:



**Figure 10 – Vitis: Successful Build**

### 5.3 Run on Hardware

With the Vivado design and Vitis application ready, the next step is to run them on the AUBoard-15P Development Kit to program the EEPROM with the desired clock configuration. Follow these steps to set up and power the board:



**Figure 11 – Board Setup**



4. Type “Y” in the serial console to confirm writing the new configuration to the EEPROM.
5. The program writes the new configuration to the EEPROM, then displays the updated memory dump and verifies its correctness. The output should look like the example in the figure below:

```

00 00 00 00 D9
Are you sure you want to proceed? (y/n): y
Proceeding with the write operation...
Current AT24AA025 Config dump:  FF FF FF FF FF FF 00 03 00 31 00 00 01 00 00 01 07 00 00 07 00 00 77
6D 00 00 00 00 00 00 00 FF FF FF FF 03 3F 00 25 00 10 00 00 00 01 00 00 D0 00 00 00 00 00 00 00 07 0
0 00 00 24 46 00 00 02 00 00 0F 00 00 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 99
99 9A 00 00 00 00 00 00 00 00 00 00 E2 0A 2B 20 00 00 00 08 00 00 00 00 00 00 00 00 00 00 27 CC 00 00 00 00
00 00 00 00 D9
Verification done: write operation succeeded

```

**Figure 14 – Programming Done**

## 6 Vivado Design for Clock Verification and Usage

In this section, we will create a Vivado hardware design to verify that the user clock, generated by the programmable clock source (**U57**), has been correctly configured and is functioning as intended. This verification step builds upon the work completed in the previous sections, where we programmed the EEPROM with the desired clock configuration using Timing Commander, Vivado, and Vitis.

The design will include:

- A basic clocking structure to drive components in the design using the user clock.
- An LED connected to the clock signal to visually confirm the clock's operation by making it blink at a frequency derived from the user clock.

By the end of this section, you will have a fully functional hardware design that validates the programmed clock and provides an example of its use.

### 6.1 Creating the Vivado design

To create a simple Vivado design that makes an LED blink using the configured user clock, follow these steps:

1. Change directory to **\$GUIDE\_HOME/** and navigate to the **clock\_verif** directory:

```

$ cd $GUIDE_HOME/
$ cd clock_verif

```

2. Open Vivado with:

```

$ vivado &

```

3. Open the **Tcl Console** at the bottom of the Vivado window and enter the following command:

```

set_param board.repoPaths ../bdf/

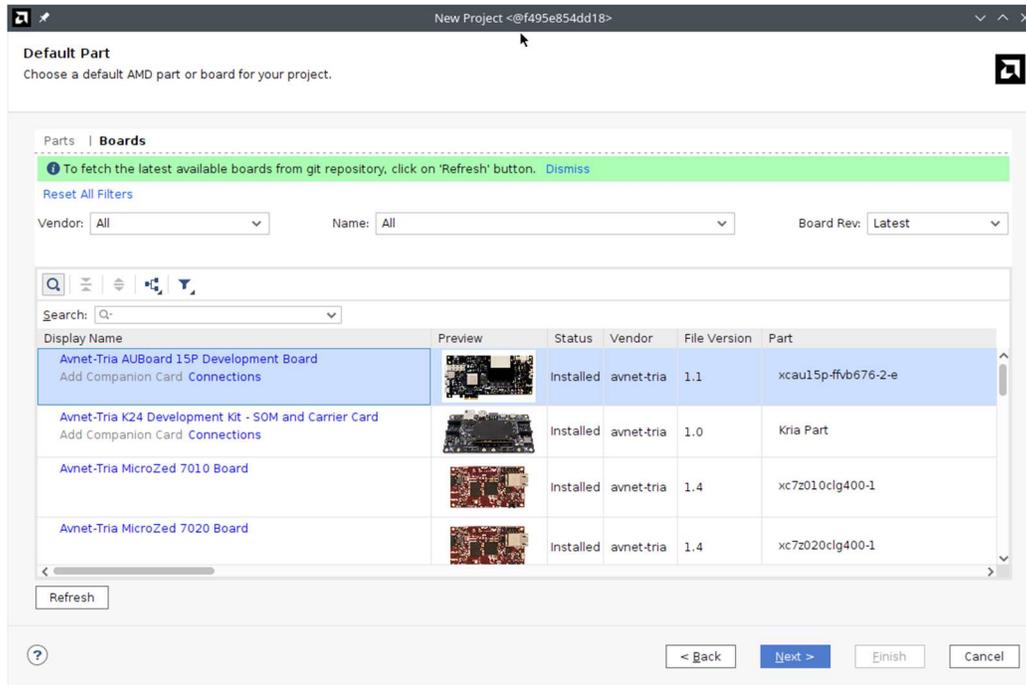
```

Adjust the **bdf** path if you cloned it to a different location during section 5.1 tasks.



**Figure 15 – Vivado: Set Repo Path**

4. Click on **FILE -> PROJECT -> NEW...** to create a new project.
5. Step Through the Wizard:
  - a. Click **NEXT** on the introduction screen.
  - b. Change the **PROJECT NAME** to **clock\_verif\_prj**.
  - c. Click **NEXT** and leave the project type as **RTL PROJECT** and enable "**Do not specify sources at this time.**"
  - d. On the **PART OR BOARD SELECTION** screen, switch to the **BOARDS** tab.
  - e. Select the **Avnet-Tria AUBoard-15P Development Board** from the list.



**Figure 16 – Vivado: Select Board Definition**

- f. Click **NEXT** and then **FINISH** to complete the project setup.
6. From the Flow Navigator, click on “**CREATE BLOCK DESIGN**” (you can leave the default name).
7. Add the following IPs to the design:
  - a. A **CLOCKING WIZARD** IP, double click on the IP to configure it.
    - In the **CLOCKING OPTIONS** tab, set the **PRIMARY SOURCE** under Input Clock Information to **GLOBAL BUFFER**.
    - In the **OUTPUT CLOCKS** tab, set the **OUTPUT FREQUENCY** of clk\_out1 to 10MHz. Uncheck the **RESET** and **LOCKED** checkboxes.



Component Name: clk\_wiz\_0

Output Clock	Port Name	Output Clocks		Phase (degrees)		Duty Cycle (%)		Drives	Matched Routing	Max Freq. of buffer
		Requested	Actual	Requested	Actual	Requested	Actual			
<input checked="" type="checkbox"/> clk_out1	clk_out1	10.000	10.00000	0.000	0.000	50.000	50.0	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	Buffer	<input type="checkbox"/>	775.194

USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Enable Optional Inputs / Outputs for MMCM/PLL:  reset  power\_down  input\_clk\_stopped

Reset Type:  Active High  Active Low

Phase Shift Mode:  WAVEFORM  LATENCY

locked  clkfbstopped

OK Cancel

**Figure 17 – Vivado: Clocking Wizard Configuration**

b. A **UTILITY BUFFER** IP:

- By default it will be an **IBUFDS** type.
- Connect its output (**IBUF\_OUT[0:0]**) to the Clocking Wizard input (**clk\_in1**).
- Right click on the input (**CLK\_IN\_D**) and click on **CREATE INTERFACE PORT**.
- Choose Slave **xilinx.com:interface:diff\_clk\_rtl:1.0** interface and name it **"user\_clock\_100mhz"**.
- Double-click the interface port and verify it is set to **100 MHz**.

c. A **BINARY COUNTER** IP:

- Configure the Output Width to 26:
- Connect the input (CLK) to the Clocking Wizard output (clk\_out1).

d. A **SLICE** IP:

- Configure Din Width = 26
- Configure Din From = 25
- Configure Din Down To = 25
- Configure Dout Width = 1

- Connect its input (**Din[25:0]**) to the Binary Counter output (**Q[25:0]**).  
Right click on the output and click on **CREATE PORT** and name it **“led”**.

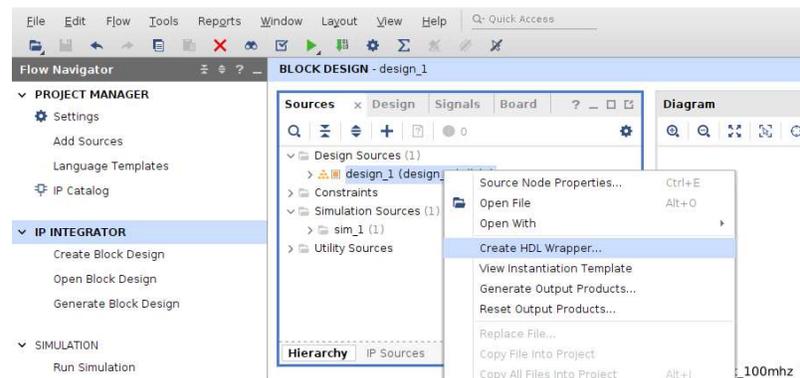
You should end up with a block design that looks like the figure shown below.



**Figure 18 – Vivado: Block Design**

This design utilizes the **user\_clock** (previously programmed to 100 MHz) as the input. A Clocking Wizard divides the frequency to **10 MHz**, which feeds a binary counter. The MSB of this counter is then used to drive the LED at a frequency that the toggling is visible.

8. In the **SOURCES** tab, right click on the “design\_1.bd” file, and select **CREATE HDL WRAPPER...**



**Figure 19 – Vivado: HDL Wrapper**

Select to let **VIVADO MANAGER WRAPPER** and **AUTO-UPDATE** and then click **OK**.

9. Click on **FILE -> ADD SOURCE**, select **ADD OR CREATE CONSTRAINTS**, and create a new file named **“aub15”**.

10. Open the **aub15.xdc** generated from the Sources tab and add the following constraints:

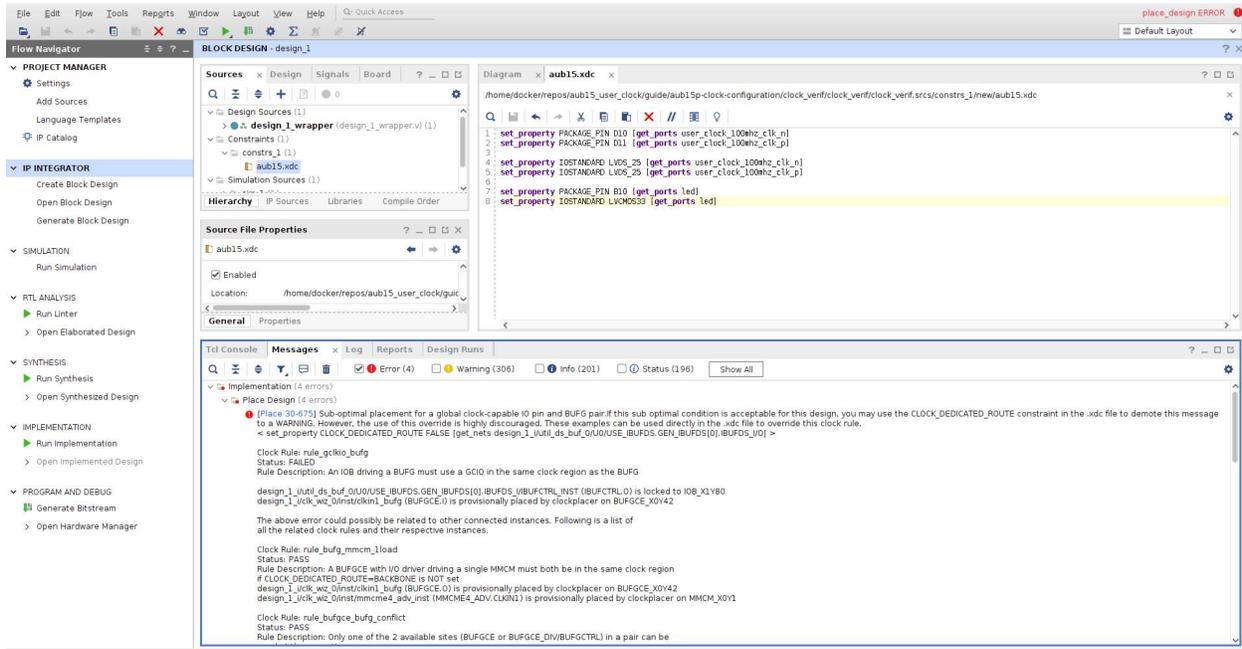
```
set_property PACKAGE_PIN D10 [get_ports user_clock_100mhz_clk_n]
set_property PACKAGE_PIN D11 [get_ports user_clock_100mhz_clk_p]

set_property IOSTANDARD LVDS_25 [get_ports user_clock_100mhz_clk_n]
set_property IOSTANDARD LVDS_25 [get_ports user_clock_100mhz_clk_p]

set_property PACKAGE_PIN B10 [get_ports led]
set_property IOSTANDARD LVCMOS33 [get_ports led]
```

**Figure 20 – Vivado: XDC Constraints**

11. Click on **GENERATE BITSTREAM** from the Flow Navigator panel. At this point the Bitstream generation should fail with this kind of error message:



**Figure 21 – Vivado: Bitstream Generation Failure**

The failure during Bitstream Generation is because the design attempts to route a clock signal through a non-dedicated clock path, which is not optimal for clock signals routed within an HDIO bank. This limitation of UltraScale+ device is described in the user guide as follows:

## Global Clock Inputs

External global user clocks must be brought into the UltraScale device on differential clock pin pairs called global clock (GC) inputs. There are four GC pin pairs in each bank that have direct access to the global clock buffers, MMCMs, and PLLs that are in the CMT adjacent to the same I/O bank. The UltraScale+ architecture has four HDGC pins per HD I/O bank. HD I/O banks are only part of the UltraScale+ family. Since HD I/O banks do not have a XIPHY and CMT next to them, the HDGC pins can only directly drive BUFGCEs (BUFGs) and not MMCMs/PLLs. Therefore, clocks that are connected to an HDGC pin can only connect to MMCMs/PLLs through the BUFGCEs. To avoid a design rule check (DRC) error, set the property `CLOCK_DEDICATED_ROUTE = FALSE`. GC inputs provide dedicated, high-speed access to the internal global and regional clock resources. GC inputs use dedicated routing and must be used for clock inputs where the timing of various clocking features is imperative. General-purpose I/O with local interconnects should not be used for clock signals.

**Figure 22 – UltraScale+ User Guide: HDIO Global Clocks**

12. To resolve this issue with routing of the clock tree, copy this constraint “**set property CLOCK\_DEDICATED\_ROUTE FALSE ...**” from the error message into the constraint file **aub15.xdc**. For reference, the updated file will look something like this:

```
set_property PACKAGE_PIN D10 [get_ports user_clock_100mhz_clk_n]
set_property PACKAGE_PIN D11 [get_ports user_clock_100mhz_clk_p]

set_property IOSTANDARD LVDS_25 [get_ports user_clock_100mhz_clk_n]
set_property IOSTANDARD LVDS_25 [get_ports user_clock_100mhz_clk_p]

set_property PACKAGE_PIN B10 [get_ports led]
set_property IOSTANDARD LVCMOS33 [get_ports led]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets
design_1_i/util_ds_buf_0/U0/USE_IBUFDS.GEN_IBUFDS[0].IBUFDS_I/O]
```

**Figure 23 – Updated XDC Constraint**

13. Save the constraints file and re-run **GENERATE BITSTREAM**. The bitstream generation should now complete successfully.

## 6.2 Running the Reference Design

After successfully generating the bitstream for the LED blinking design, the next step is to load the bitstream onto the target board and verify its functionality. Follow the steps below to program the reference design into the target hardware:

1. Prepare the Hardware
  - a. Connect the 12V power supply to the power connector (**J51**) on the board.
  - b. Use a USB-micro-USB cable to connect your computer to the JTAG UART port (**J9**) on the board.
  - c. Turn on the board using the power switch (**SW1**).
2. Program the FPGA
  - a. From the Vivado **FLOW NAVIGATOR**, click on **OPEN HARDWARE MANAGER**.
  - b. Click on **OPEN TARGET**, then **AUTO CONNECT** to connect to the target hardware.
  - c. Ensure the AUBoard-15P Development Kit is listed in the connected devices.
  - d. In the Hardware Manager, click on **PROGRAM DEVICE** and select the target FPGA.
  - e. In the file browser that appears, navigate to your generated bitstream file (e.g., **design\_1\_wrapper.bit**)
  - f. Click **PROGRAM**.

- g. Once the programming process is complete, the hardware will be configured with the LED blinking design.

### 3. Verify the LED Blinking

Observe the onboard D32 LED, as shown in the picture above:



**Figure 24 – Board: D32 LED Blinking**

It should now blink with a period of approximately 6.71 seconds, as configured in the design. This 6.71-second period results from the 100 MHz user clock being divided down to 10 MHz by the Clocking Wizard, and then further divided by  $2^{26}$  through the binary counter, making the LED toggle with a 6.71s period.

## 7 Conclusion

This concludes the Configuring the Clock Generator Reference design for the AUBoard-15P Development Kit. The final LED blinking verification is a practical test to ensure the successful programming of the EEPROM and the proper configuration of the user clock, which aligns with the overall project goal of enabling the end user to program the out-of-box clock configuration or to potentially allow the end-user to customize the clock configuration to better suit their application needs.

## 8 Getting Help and Support

If additional support is required, TRIA Technologies has many avenues to search depending on your needs.

For general question regarding AUBoard-15P Development Kit, please visit our website at <http://avnet.me/auboard-15p>. Here you can find any available documentation, technical specifications, videos and tutorials, reference designs and other support.

Detailed questions regarding AUBoard-15P Development Kit hardware design, software application development, using AMD tools, training and other topics can be posted on the AUBoard-15P Development Kit Support Forum at <http://avnet.me/auboard-15p-forum>. Avnet's technical support team monitors the forum during normal business hours in North America.

Those interested in customer-specific options on AUBoard-15P Development Kit can send inquiries to [customize@avnet.com](mailto:customize@avnet.com).