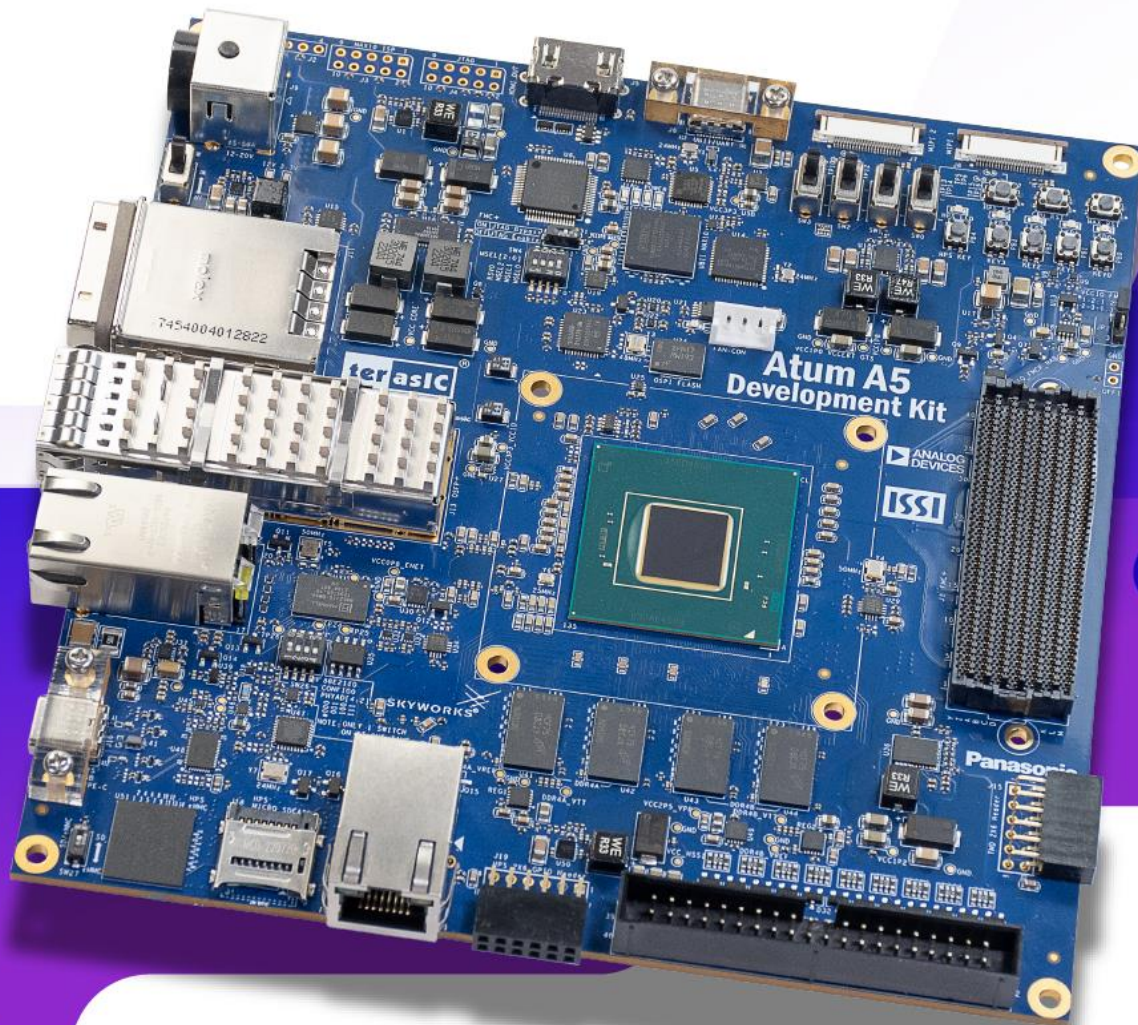


ATUM A5 Board

Demonstration Manual



FPGA

Contents

| | | |
|------------------|--|-----------|
| Chapter 1 | Overview..... | 4 |
| Chapter 2 | Examples For FPGA | 5 |
| 2.1 | Basic Nios V control demo for Temperature/ Power/ Fan..... | 5 |
| 2.2 | Board Information IP | 8 |
| 2.3 | Si5391B IP | 12 |
| 2.4 | Clock Controller demo for Si5391B | 19 |
| 2.5 | HDMI TX | 21 |
| 2.6 | Low Latency 40G Ethernet Example | 25 |
| 2.7 | DDR4 Test by Test Engine IP | 29 |
| Chapter 3 | Examples for HPS SoC..... | 33 |
| 3.1 | HPS LED/KEY..... | 33 |
| 3.2 | Network Socket | 37 |
| 3.3 | Build C/C++ Project..... | 44 |
| Chapter 4 | Transceiver Verification | 46 |
| 4.1 | Transceiver Test Code | 46 |
| 4.2 | Loopback Fixture..... | 46 |
| 4.3 | Testing by Transceiver Test Code | 47 |



| | | |
|------------------|--------------------------------------|-----------|
| Chapter 5 | <i>Additional Information</i> | 53 |
| 5.1 | Getting Help | 53 |

Chapter 1

Overview

This Manual will introduce the various application demonstrations on **Atum A5 Development Kit**. These demonstrations cover most of the interfaces on the board. Let users familiarize using these interfaces of the board. Demonstrations according to FPGA fabrics and HPS are divided into three categories:

- **Pure use of FPGA fabric resources (Chapter 2)**
- **Pure use of HPS fabric resources (Chapter 3)**

Finally, to complete the following demonstration, user needs to install the following software in the computer:

- [Intel Quartus® Prime Pro Edition Software Version 24.3](#) or later.
- [Intel SoC Embedded Design Suite\(EDS\) Professional Edition](#)

Note: To run the demo bath file with the Nios II CPU of the demonstration on windows system, user need to install the Windows Subsystem for Linux (WSL) first then you can run the batch file. Please refer to the link to install : [Getting Start Install WSL](#)

Chapter 2

Examples For FPGA

This chapter provides examples of advanced designs implemented by RTL or Qsys on the **Atum A5 Development Kit**. These reference designs cover the features of peripherals connected to the FPGA, such as DDR4, temperature monitor, PLL clock setting and Power monitor. All the associated files can be found in the directory **\Demonstrations\FPGA** of Atum A5 System CD.

2.1 Basic Nios V control demo for Temperature/ Power/ Fan

This demonstration shows how to use the Nios V processor to measure the power consumption based on the built-in power measure circuit. The demonstration also includes a function of monitoring system temperature with the on-board temperature sensor and monitoring fan rotation speed.

■ System Block Diagram

Figure 2-1 shows the system block diagram of this demonstration. The 12V input power monitor, temperature sensor and fan controller connected to the system MAX10 FPGA and controlled by internal logic circuits. All collected status data or control commands will be sent to the SPI slave block so that the Agilex FPGA can read it through the SPI interface.

In the Agilex FPGA, an SPI master IP (implemented by HDL) will read these external sensor data from the MAX10 FPGA through SPI interface. The Nios system will read these information through PIO controllers.

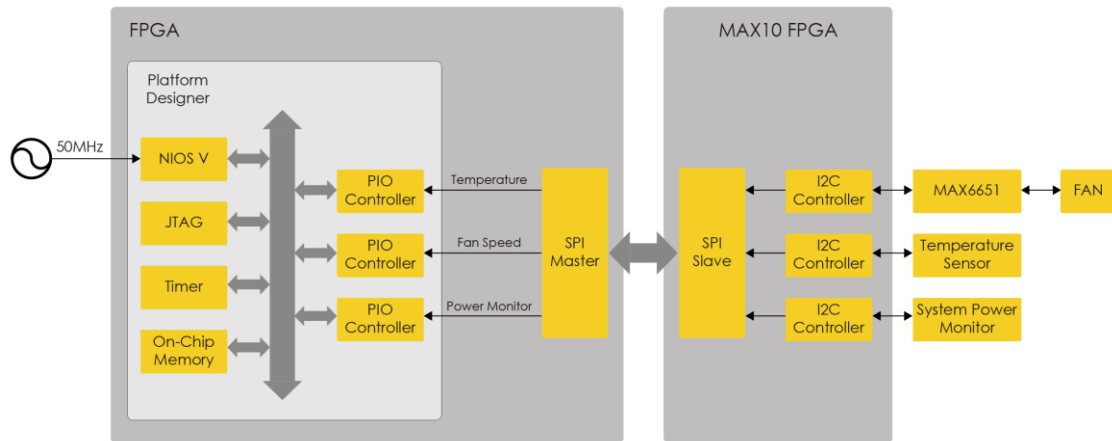


Figure 2-1 Block Diagram of the Nios V Basic Demonstration

The system provides a menu in command line window, as shown in **Figure 2-2** to provide an interactive interface. With the menu, users can perform the test for the board info sensor. Note, pressing 'ENTER' should be followed with the choice number.

```

Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x29c lma 0x20
Loading section .text, size 0x21904 lma 0x2bc
Loading section .rodata, size 0x1590 lma 0x21bc0
Loading section .rwdata, size 0x1980 lma 0x24ad0
Start address 0x000008c4, load size 150224
Transfer rate: 69 KB/sec, 11555 bytes/write.
[Inferior 1 (Remote target) detached]
INFO: Sending resume to OpenOCD.
===== Agilex Demo Program =====
[0] Display Board Info
Input your choice:0
==== Temperature ====
      FPGA: 39*C
      Board 1: 35*C
      Board 2: 41*C
      Power: 45*C

==== Fan ====
      Fan RPM: 3420

==== Power (12V) Monitor ====
      Voltage      = 12.195 V
==== Core Power Monitor ====
      Voltage      = 0.798 V
      Current      = 1.250 A
      Power        = 0.997 W
==== VCCIP2 Power Monitor ====
      Voltage      = 1.195 V
      Current      = 0.310 A
      Power        = 0.370 W
Display Board Info Test:PASS
===== Agilex Demo Program =====
[0] Display Board Info
Input your choice:

```

Figure 2-2 Menu of Demo Program

In board info test, the program will display local temperature, remote temperature, 12V

input power monitor and fan rotation speed. The remote temperature is the FPGA temperature, and the local temperature is the board temperature where the temperature sensor located. The board also provides circuitry to monitor important voltages, currents and power consumption in real time.

■ Demonstration File Location

- Hardware project directory: Board_Info
- Bitstream used: Board_Info.sof
- Software project directory: Board_Info\software
- Demo batch file: Board_Info\demo_batch\test.bat

■ Install Ashling RiscFree IDE

Before executing this demo with RISC-V core, users need to install Ashling RiscFree IDE for Intel® FPGAs to ensure that this batch file can be executed correctly. The file name should be *RiscFreeSetup-<Quartus Version>-windows.exe*. Users can find it under the [Quartus Pro download page](#) (Individual Files tab).

■ Demonstration Setup and Instructions

1. Make sure Quartus Prime is installed on the Host PC.
2. Power on the FPGA board.
3. Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
4. Execute the demo batch file “test.bat” under the batch file folder: Board_Info\demo_batch.
5. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in command line window.
6. For temperature, power monitor and fan test, please input key ‘0’ and press ‘Enter’ in the command line window, as shown in **Figure 2-3**.


```

Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x29c lma 0x20
Loading section .text, size 0x21904 lma 0x2bc
Loading section .rodata, size 0x1590 lma 0x21bc0
Loading section .rwddata, size 0x1980 lma 0x24ad0
Start address 0x000008c4, load size 150224
Transfer rate: 69 KB/sec, 11555 bytes/write.
[Inferior 1 (Remote target) detached]
INFO: Sending resume to OpenOCD.
===== Agilex Demo Program =====
[0] Display Board Info
Input your chioce:0
==== Temperature ====
      FPGA: 39*C
      Board 1: 35*C
      Board 2: 41*C
      Power: 45*C

==== Fan ====
      Fan RPM: 3420

==== Power (12V) Monitor ====
      Voltage      = 12.195 V
==== Core Power Monitor ====
      Voltage      = 0.798 V
      Current      = 1.250 A
      Power        = 0.997 W
==== VCC1P2 Power Monitor ====
      Voltage      = 1.195 V
      Current      = 0.310 A
      Power        = 0.370 W
Display Board Info Test:PASS
===== Agilex Demo Program =====
[0] Display Board Info
Input your chioce:

```

Figure 2-3 Board Info Demo

2.2 Board Information IP

This section will introduce an IP which can be placed in the Agilex FPGA and allows users to obtain board status information such as power, temperature status and fan speed on the ATUM A5 board.

The ATUM A5 board provides several sensors to monitor the status of the board, such as FPGA temperature, board power monitor, and fan speed status. These interfaces are connected to the system MAX FPGA on the board. The logic in the system MAX FPGA will automatically read the status values of these sensors and store them in the internal register. As shown in **Figure 2-4**, there is an SPI slave IP in the system MAX FPGA will read the value of the board status from these registers and it can be output to SPI master logic via SPI interface.

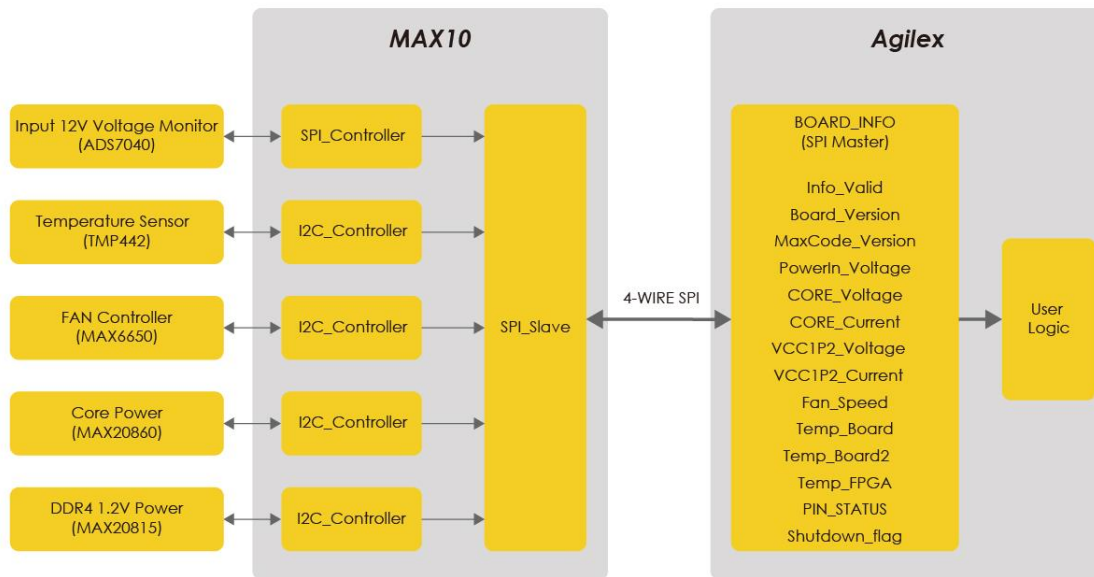


Figure 2-4 Block diagram of the fan speed control demonstration

User can placing a board information IP (BOARD_INFO.v ; spi master) provided by Terasic in the Agilex FPGA, the board status can be obtained via SPI interface from the system MAX FPGA and output to user logic.

The board information IP can be obtained from the following path in the system CD:
Demonstration/FPGA/Board_info_RTL/board_information_ip/BOARD_INFO.v

Figure 2-5 shows the input and output pins of the board information IP. Detailed pin descriptions and functions can be obtained from **Table 2-1** Board information IP input and output ports. The user only needs to provide the IP 50Mhz clock and the reset control signal. The IP will automatically communicate with the system MAX FPGA to get the board status value via the SPI interface. When the logic level of the Info_Valid signal is from low to high, it means that the board status has been updated and can be used.

Finally, **Figure 2-6** shows the status of the IP during execution.

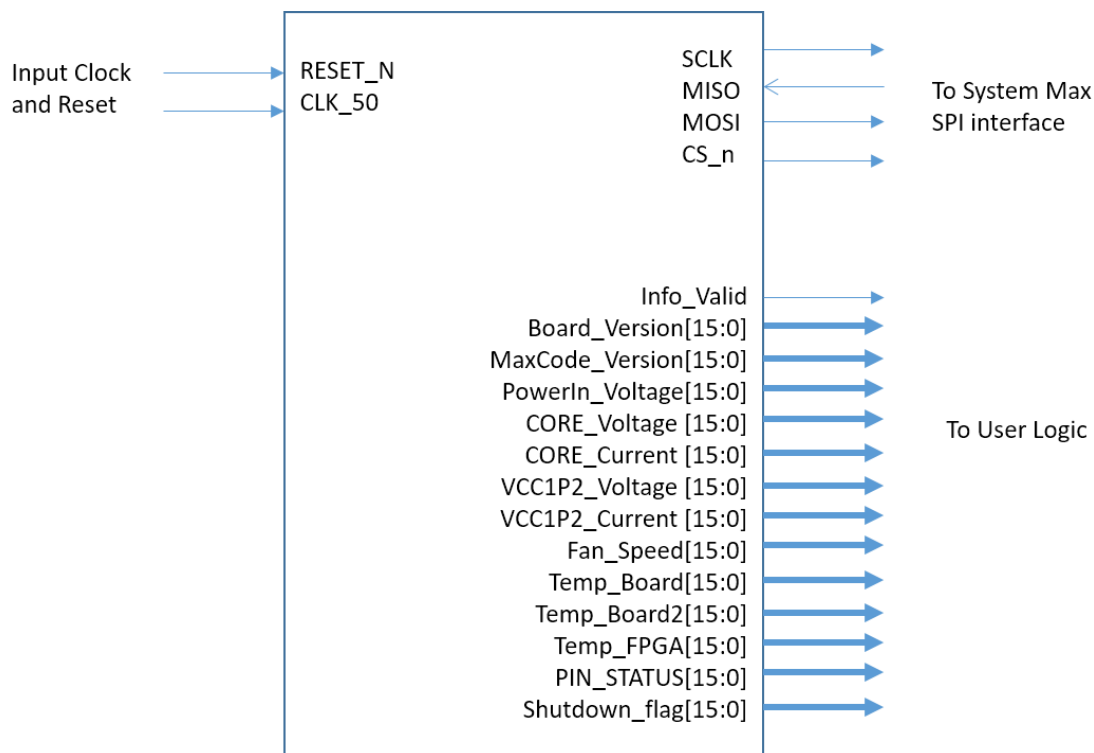


Figure 2-5 Pin out of the board information IP

Table 2-1 Board information IP input and output ports

| Port Name | Direction | Width(Bit) | Description |
|---------------|-----------|------------|---|
| CLK_50 | Input | 1 | Clock input for IP, please input 50Mhz clock. |
| RESET_N | Input | 1 | Reset signal for IP, reset all logic. |
| MOSI | Output | 1 | Master data output. Please connect this signal to the INFO_SPI_MOSI pin. |
| MISO | Input | 1 | Master data input. Please connect this signal to the INFO_SPI_MISO pin. |
| CS_n | Output | 1 | Slave Select, Master output. Please connect this signal to the INFO_SPI_CS_n pin. |
| SCLK | Output | 1 | Serial Clock, SPI master output to slave. Please connect this signal to the INFO_SPI_SCLK pin. |
| Info_Valid | Output | 1 | Information valid, logic high indicates board status updated ready. |
| Board_Version | Output | 16 | This information indicates the version of the ATUM A5 board. It will be started at 0x000A. |

| | | | |
|-----------------|--------|----|--|
| MaxCode_Version | Output | 16 | This information indicates the version of the System MAX 10 FPGA code. It will be started at 0x0001. |
| PowerIn_Voltage | Output | 16 | 12V Voltage, the unit of the output value is mV. If the PowerIn_Voltage output value is "12050" that means 12.05V for 12V power |
| CORE_Voltage | Output | 16 | Core voltage of the first power channel , Unit is mV |
| CORE_Current | Output | 16 | Current of the first power channel , Unit is 10mA |
| VCC1P2_Voltage | Output | 16 | VCC1P2 (DDR4) voltage of the first power channel , Unit is mV |
| VCC1P2_Current | Output | 16 | VCC1P2 (DDR4) current of the first power channel , Unit is 10mA |
| Fan_Speed | Output | 16 | First fan speed of the board. The unit of the output value is RPM. |
| Temp_Board | Output | 16 | First ambient temperature of the development board. The unit of the output value is Celsius. |
| Temp_Board2 | Output | 16 | Second ambient temperature of the development board. The unit of the output value is Celsius. |
| Temp_FPGA | Output | 16 | Core FPGA temperature of the development board. The unit of the output value is Celsius. |
| Shutdown_flag | Output | 16 | BIT5~15 : Reserved to 0. BIT4: 1 ,when CORE IC Temperature $\geq 95^{\circ}\text{C}$ BIT3: 1 ,when CORE_Current $\geq 35\text{A}$ BIT2: 1 ,when FPGA Temperature $\geq 95^{\circ}\text{C}$ BIT1: 1 ,when Board1 Temperature $\geq 95^{\circ}\text{C}$ BIT0: 1 ,when Board Temperature $\geq 95^{\circ}\text{C}$ |
| PIN_STATUS | Output | 16 | BIT8~15 : Reserved to 0. BIT7: FAN_ALERT_n , When the fan speed is abnormal, this bit is 0. BIT6: Reserved to 1. BIT5: When shutdown occurs, this bit is 0. BIT4: Reserved to 1. |

| | | | |
|--|--|--|---|
| | | | BIT3: Reserved to 0. BIT2: FPGA_CONF_DONE ,FPGA Configure success, this bit is 1. BIT1: Reserved to 1. BIT0: Reserved to 1. |
|--|--|--|---|

| | | |
|------------------------------------|--|-------|
| BOARD_INFO_ Board_Version[15..0] | | 000Bh |
| BOARD_INFO_ MaxCode_Version[15..0] | | 2 |
| BOARD_INFO_ PowerIn_Voltage[15..0] | | 12316 |
| BOARD_INFO_ CORE_Voltage[15..0] | | 796 |
| BOARD_INFO_ Fan_Speed[15..0] | | 3420 |
| BOARD_INFO_ Temp_FPGA[15..0] | | 41 |
| BOARD_INFO_ Info_Valid | | |

Figure 2-6 Waveform of the board status output

2.3 Si5391B IP

There is a SKYWORKS Si5391B clock generator on the Atum-A5 board can provide adjustable frequency reference clock (See [Figure 2-7](#) for XCVR 1B/4A/4B). The Si5391B clock generator can output differential frequencies though I2C interface configuration. This section will show you how to use FPGA RTL IP to configure each Si5391B PLL and generate users desired output frequency to each peripheral.

The Si5391B control IP is located in the System CD folder:

"\\Demonstrations\FPGA\si5391B_clock_controller\Si5397B_IP"

Developers can use the IP directly in their Quartus top. Developers can refer to the example in Demonstrations/Si5391B_Clock_Controller folder. This example shows how to instantiate the IP in Quartus top project.

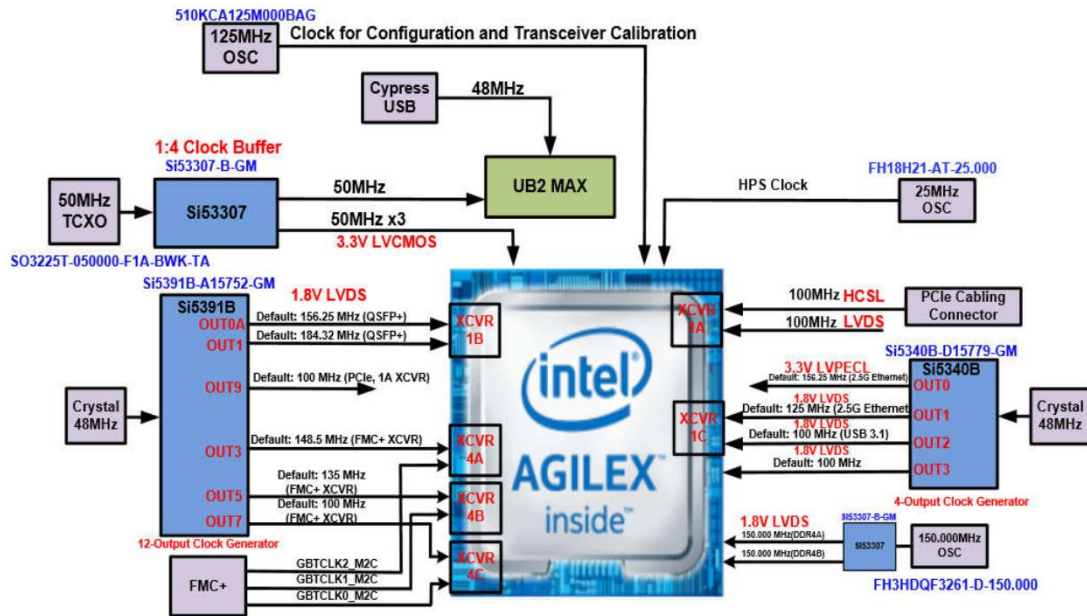


Figure 2-7 Clock tree of the Atum-A5 Board

■ Si5391B IP Block Diagram

Figure 2-8 shows the Block Diagram of Si5391B IP. The Si5391B configuration data is stored in a ROM. The ROM contains register data to configure Si5391B via I2C interface. The register data original come from SKYWORKS ClockBuilder Pro Utility. The ROM content is initialized by a Memory Initialization File (.mif) file. The IP is located in the System CD folder:

CD\Demonstration\FPGA\Si5391B_Clock_Controller\Si5391B_IP

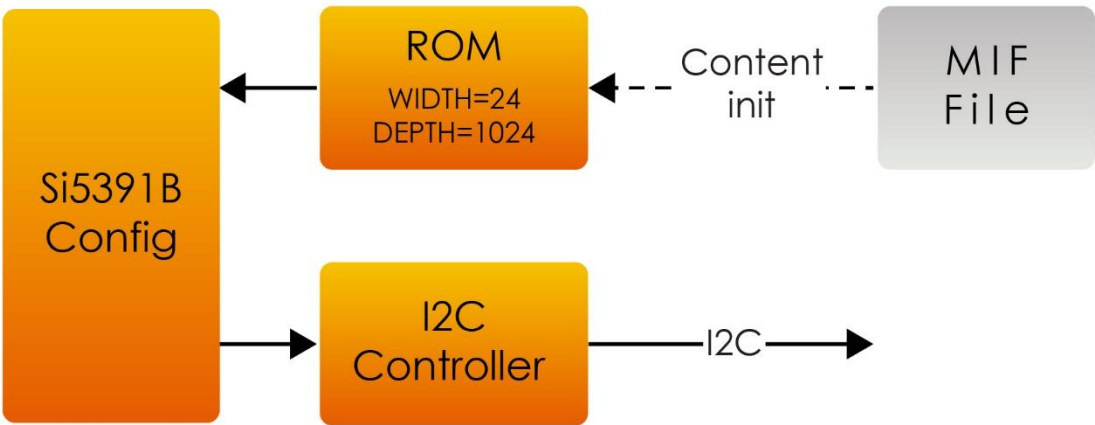


Figure 2-8 Si5391B IP Block Diagram

■ .MIF File Generation

Figure 2-9 shows the flow to generate the .mif Memory Initialization File which is used to initialize ROM in Si5391B IP. First, users have to download SKYWORD ClockBuilder Pro software utility. Then launch ClockBuilder Pro and open Terasic Si5391B golden project. In ClockBuilder Pro, users modify desired output frequency and export register setting file (.txt). Finally, use Terasic ROM_MIF.exe tool to convert the register setting file (.txt) to a ROM initialization file .mif. The ROM_MIF.exe tool and Si5391B golden project are located in the System CD folder:

CD\Demonstration\FPGA\Si5391B_Clock_Controller\TXT2MIF

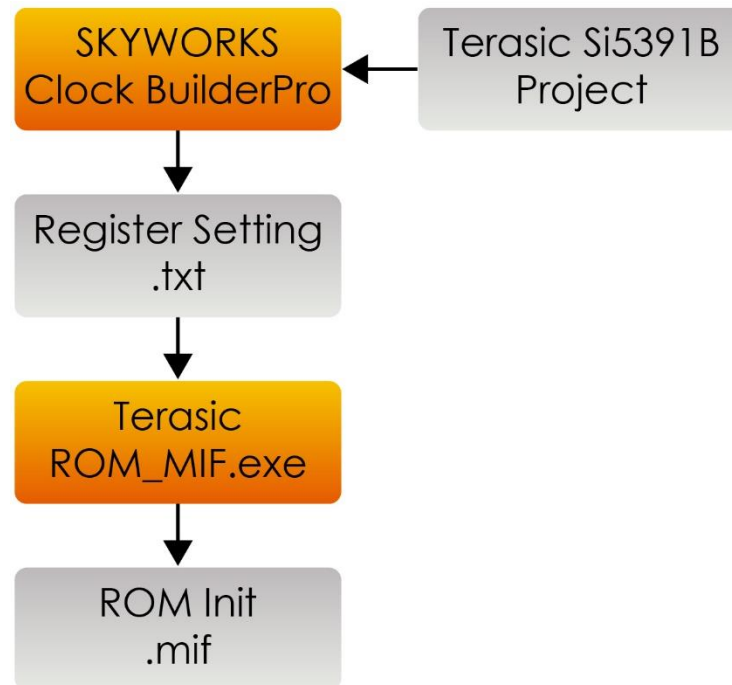


Figure 2-9 MIF Generation Flow

■ Step by Step to Update Register ROM in Si5391B IP

If the Si5391B built-in frequencies are not users' desired, users can refer to the below steps to the modify register parameter settings (stored in ROM) to make Si5391B to output desired frequencies.

1. Firstly, download ClockBuilder Pro Software (See **Figure 2-10**), which is provided by SKYWORKS. This tool can help users to set the Si5391B's output frequency of each channel through the GUI interface, and it will automatically calculate the Register parameters required for each frequency. ClockBuilder Pro 4.11 is recommend. The tool download link:

[http://url.terasic.com/clockbuilder ro ofware](http://url.terasic.com/clockbuilder_roftware)



Figure 2-10 ClockBuilder Pro Wizard

2. Launch ClociBuilder Pro, click “Open Project” to open Terasic Si5391B Golden project Terasic_Si5391B.slabtimeproj as shown in **Figure 2-11**. The Si5391B project is located in the system CD folder:

CD\Demonstration\FPGA\Si5391B_Clock_Controller\TXT2MIF

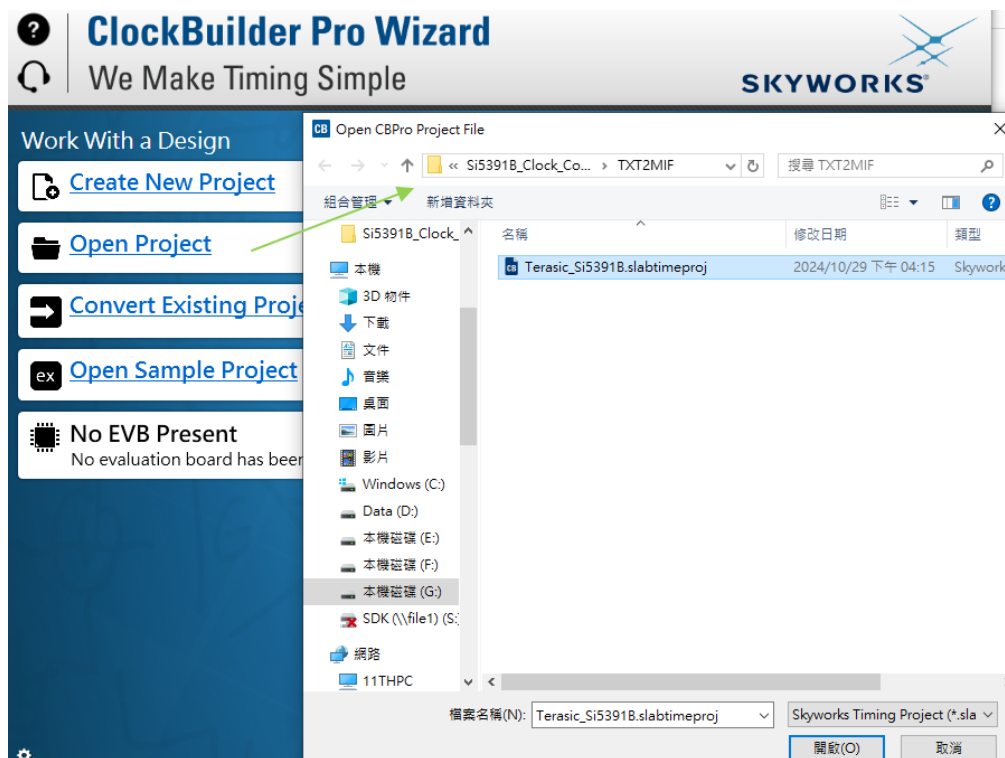


Figure 2-11 Open Project in CBPro

- As shown in **Figure 2-12**, click “Outputs” to open Output Clocks page.

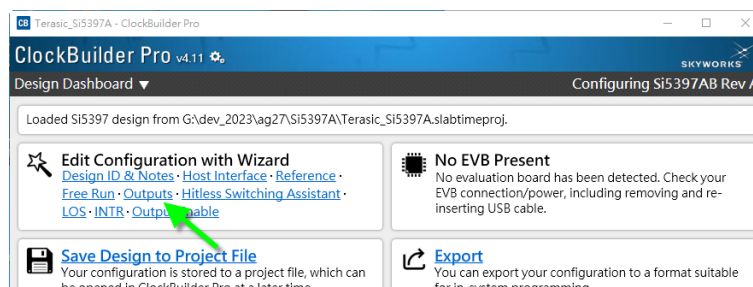


Figure 2-12 Click “Outputs” in CBPro

- In the Output Clocks page, specified desired output frequencies and click “Finish” button as shown in **Figure 2-13**.

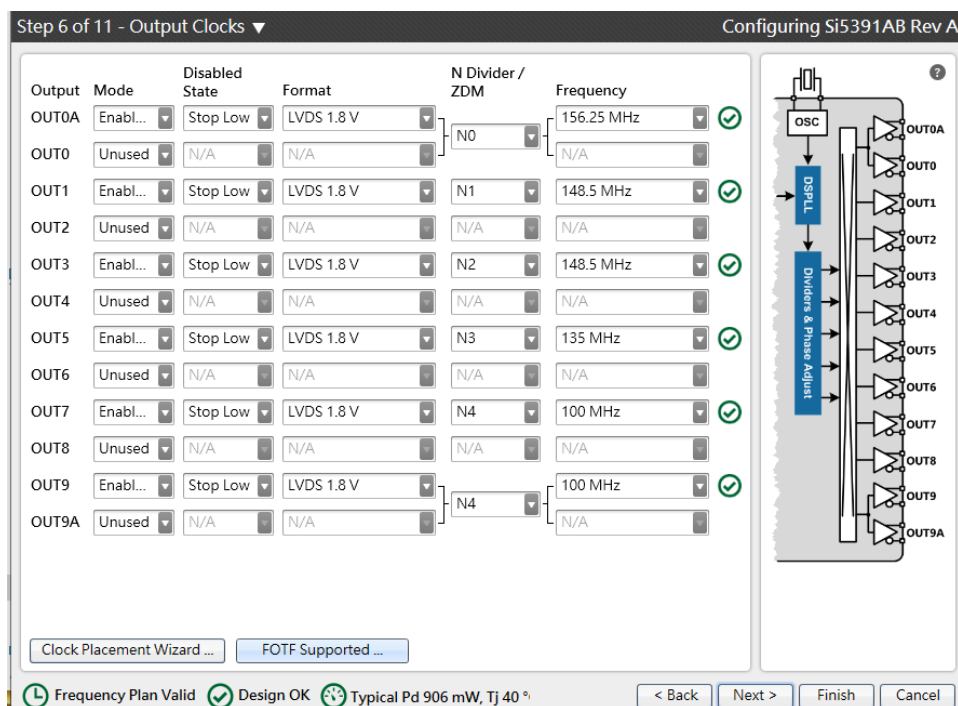


Figure 2-13 Specify Output Clock Frequencies on CBPro

- As shown in **Figure 2-14**, click “Export” in ClockBuilder Pro to open Si5391 Export Dialog.

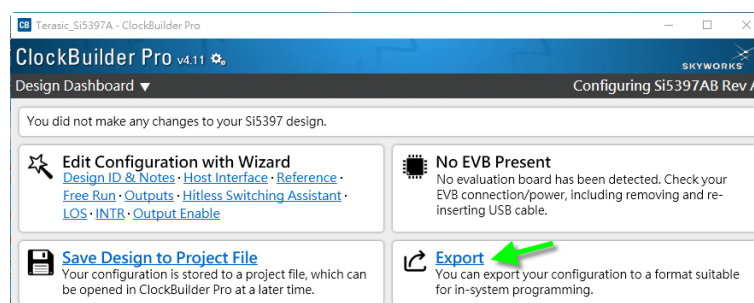


Figure 2-14 Click “Export” in CBPro

- In the Si5391 Export dialog, select “Register File” tab. Then, select “Comma Separated Values (CSV) File”, and check both items “include summary header” and “include pre- and post-write control register writes” as shown in **Figure 2-15**. Finally, click “Save to File...” button to save file as **Si5391B-Registers.txt**.

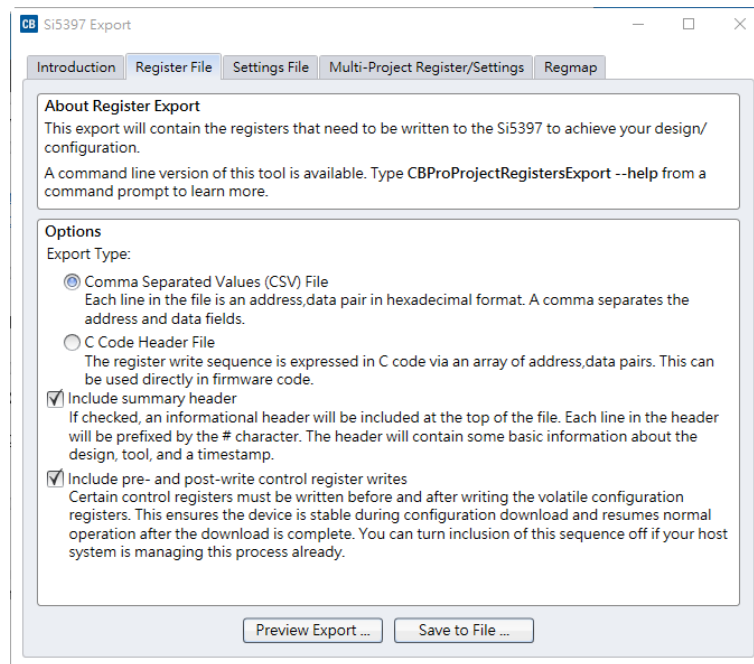


Figure 2-15 Save Register Files

7. Launch Teraisc ROM_MIF utility as shown in **Figure 2-16**. Click “Generate .MIF” button and select **Si5391B-Registers.txt** file generated in previous step, then **Si5391B-Registers.mif** is generated.

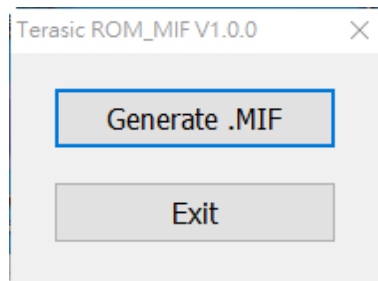


Figure 2-16 Teraisc ROM_MIF utility

8. Use Quartus Pro to open ROM IP file (si5391B_rom.ip) in Si5391B IP. Select “Mem Init” tab, and click file name browse button to select the previous generated file **Si5391B-Registers.mif**. Then, click “Generate HDL...” button to regenerate ROM IP with new content. The ROM IP file is located in the System CD folder:

CD\Demonstration\FPGA\Si5391B_Clock_Controller\Si5391B_IP

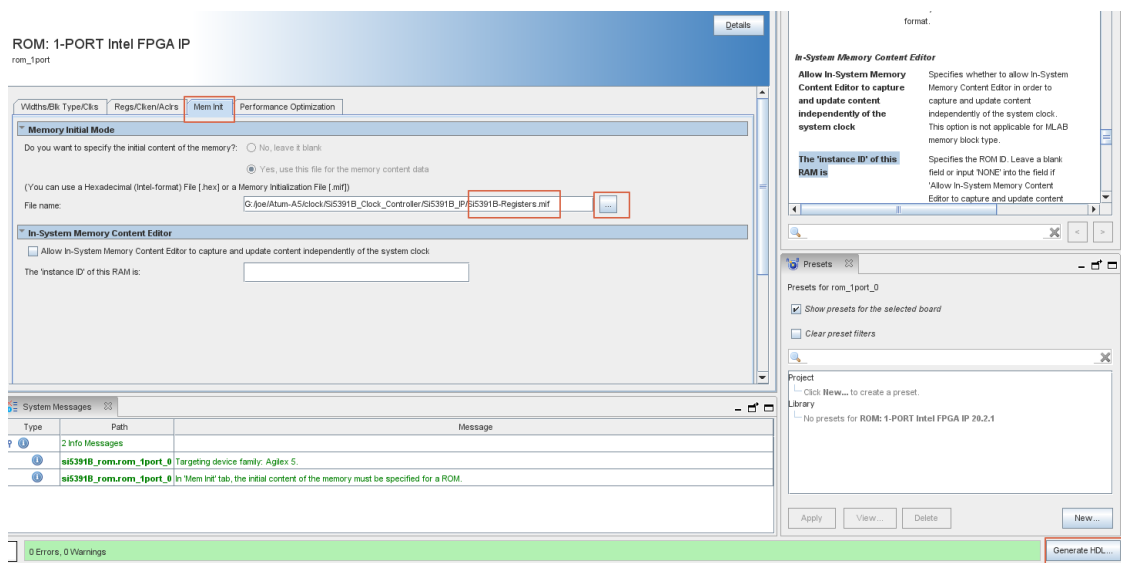


Figure 2-17 .mif file setting in ROM IP

2.4 Clock Controller demo for Si5391B

This demonstration shows how to use SI5391B Control IP (written by Verilog) to configure SI5391B clock generator to generate the desired frequency, and using Quartus SignalTap II tool to verify whether the output frequency of SI5391B is correct.

■ System Block Diagram

Figure 2-18 shows the system block diagram of this demonstration. The Si5391B clock generator is controlled by SI5391B IP through the I2C bus. In this demonstration, Si5391B OUT1 (map to CIPRI_REFCLK_p) frequency is changed from 184.32Mhz to 148.5Mhz when users press the FPGA User Button[0] to active the Si5391B reconfigure process. Press FPGA User Button[1] will reset Si5391B, and the Si5391B will output default frequencies.

A clock frequency measurement module is used to measure the frequencies of CIPRI_REFCLK_p input clocks. The measured frequencies are displayed in SignalTap II tool. Users can use the SignalTape II to observe the measured clock frequency to verify whether the Si5391B output frequency is correct

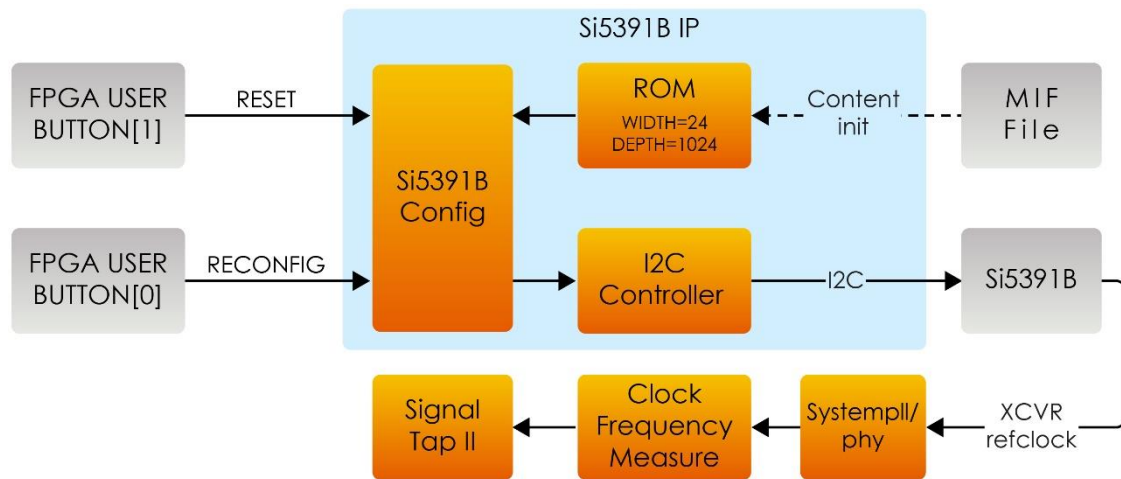


Figure 2-18 Block Diagram of the Si5391B Clock Controller Demonstration

■ Design Tools

- Quartus Prime 24.2 Pro Edition

■ Demonstration File Locations

- Hardware project directory: Si5391B_Clock_Controller
- Bitstream used: golden_top.sof
- Demo batch file: Si5391B_Clock_Controller\demo_batch\test.bat

■ Demonstration Setup and Instructions

- Make sure Quartus Pro is installed on the host PC.
- Power on the FPGA board.
- Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
- Execute the demo batch file “test.bat” under the batch file folder: Si5391B_Clock_Controller \demo_batch
- Using Quartus II to open the SignalTab II file “stp1.stp” under the batch file folder: Si5391B_Clock_Controller \demo_batch.
- Switch to the SignalTab II window to observe the clock registers and verify the frequency. CIPRI_REFCLK_p shows default output clock frequency 184,320,000 Hz, as shown in Figure 2-19.

| | | |
|---------|-----------------------------------|-----------|
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p0 FREQ[31..0] | 148498300 |
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p1 FREQ[31..0] | 134998455 |
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p2 FREQ[31..0] | 99998855 |
| Pre-Syn | ⊕ fcipri_xcvr_ref_clk FREQ[31..0] | 184317890 |
| Pre-Syn | ⊕ fpcie_xcvr_ref_clk FREQ[31..0] | 99998855 |

Figure 2-19 SignalTab II shows 184.32Mhz

- Press FPGA User Button[0], to start Si5391 reconfigure process. CIPRI_REFCLK_p shows 148,500,000 Hz as shown in Figure 2-14.

| | | |
|---------|-----------------------------------|-----------|
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p0 FREQ[31..0] | 148498267 |
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p1 FREQ[31..0] | 134998424 |
| Pre-Syn | ⊕ fMCP_GBTCLK_M2C_p2 FREQ[31..0] | 99998832 |
| Pre-Syn | ⊕ fcipri_xcvr_ref_clk FREQ[31..0] | 148498267 |
| Pre-Syn | ⊕ fpcie_xcvr_ref_clk FREQ[31..0] | 99998832 |

Figure 2-20 SignalTab II shows 148.5Mhz

- Press FPGA User Button[1] to reset Si5391. CIPRI_REFCLK_p shows 184,320,000 Hz as shown in Figure 2-13

2.5 HDMI TX

This section gives instructions to program the HDMI transmitter to generate video pattern and audio source. The entire reference is composed into three parts: video design, audio design, and I2C design. A set of built-in video patterns and audio serial data will be sent to the HDMI transmitter to drive the HDMI display with speaker. Users can hear the beeping sound from the speaker. The resolution in this demo is set to 1080p @60Hz.

■ System Block Diagram

Figure 2-21 shows the system block diagram of this reference design. The HDMI Transmitter is configured via I2C interface by I2C Controller and I2C HDMI Config. It is necessary to configure the HDMI transmitter according to the desired settings.

An interrupt mechanism called Hot Plug Detect (HPD) is implemented in I2C HDMI config. to re-configure HDMI transmitter when HPD interrupt occurs.

The Video Patter Generator was designed to send video patter to HDMI transmitter. The Audio Generator were designed to send audio pattern to HDMI transmitter. The audio is transmitted via I2S interface in this demo.

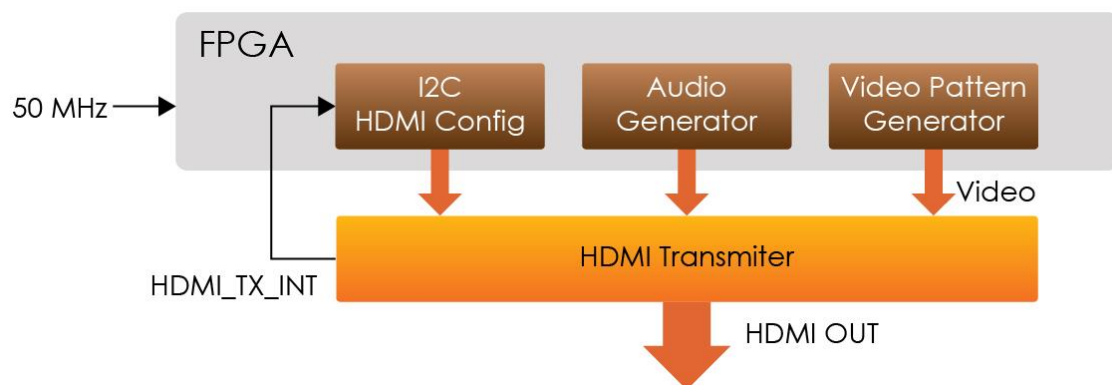


Figure 2-21 Block Diagram of the HDMI TX Demonstration

■ Register of HDMI Transmitter (ADV7513)

Users can save lots of developing time by paying attention to the settings of video format and audio frequency in register at address 0x15 and the format of register at address 0xAF prior to the development of HDMI transmitter. This demo uses 48KHz sampling rate and the video format is 24-bit RGB 4:4:4. For more details, please refer to the document ADV7513_Programming_Guide_R0.pdf.

■ Audio Generator

The ADV7513 can accommodate 2 to 8 channels of I2S audio at up to a 192 KHz sampling rate. The ADV7513 supports I2S standard, left-justified serial audio, and right-justified serial audio. **Figure 2-22** shows the left-justified serial audio with I2S standard audio of 16-bit per channel.

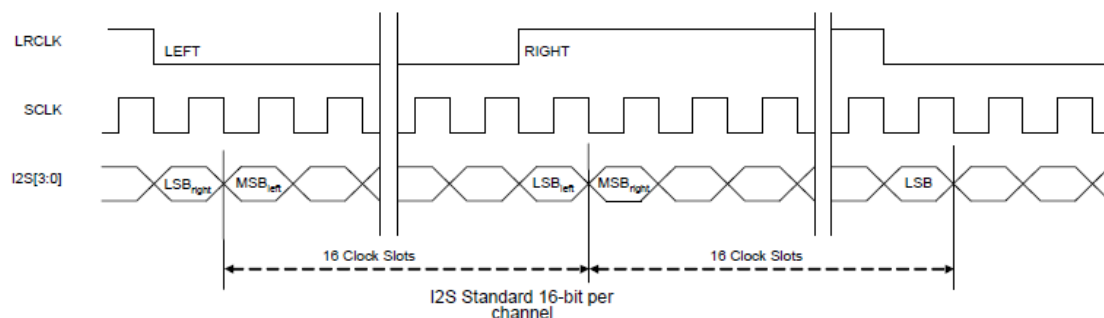


Figure 2-22 I2S standard audio with 16-bit per channel

If users want to modify the frequency of audio output, the register value at register 0x15 has to be modified according to the document ADV7513_Programming_Guide_R0.pdf. The I2S standard uses MSB to LSB serial way of transmitting. This demo uses sinusoid signal to synthesis sound from a reference of look up table created by calculated sinusoid wave data.

■ Video Pattern Generator

The module "Video Pattern Generator" copes with generating video patterns to be presented on the LCD monitor. The pattern is composed in the way of 24-bit RGB 4:4:4 (RGB888 per color pixel without sub-sampling) color encoding, which corresponds to the parallel encoding format defined in **Table 2-2** of the "ADV7513 Hardware User's Guide," as shown below.

Table 2-2 Display modes of the HDMI TX demonstration

| Pixel Data [23:0] | | | | | | | | | | | | | | | | | |
|-------------------|----|----|----|----|----|--------|----|---|---|---|---|--------|---|---|---|---|---|
| 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| R[7:0] | | | | | | G[7:0] | | | | | | B[7:0] | | | | | |

■ Demonstration File Locations

- Hardware project directory: \HDMI_TX
 - Bitstream used: golden_top.sof
- Demo batch file : \HDMI_TX\demo_batch\test.bat

■ Demonstration Setup and Instructions

- Please make sure both Quartus II and USB-Blaster II driver are installed on the host PC.
- Connect the Atum A5 board to the LCD monitor through a HDMI cable.
- Power on the board.
- Launch the "test.bat" batch file from the "\HDMI_TX\demo_batch" folder. After the programming and configuration are successful, the screen should look like the one shown in **Figure 2-23** .

```

G:\tmp\ax5soc\demo\c>H:\intel\FPGA_pro\24.1\quartus\bin64\quartus_pgm.exe -m jtag -c 1 -o "p:golden_top.sof@1"
Info (19848): Regular SEL info => 9 sector(s), 9 thread(s), 10000000 interval time in microsecond(s)
Info (19848): IO hash is 2B3F04DF9BE17E1964750C1A0B079741B107B51C898FB32F24979E421010F90
Info (19848): Keyed hash is D9649D6992F6CEE3C79915B1A75A7FC5FB2BAEE203EDE0DD253A79B4E40E48CE
Info (19848): Design hash is 55EB640C1234024FFC777848062F5B0BF79CF0231AA57BF763E1C03857BE26EF
Info (19848): IO hash is 2B3F04DF9BE17E1964750C1A0B079741B107B51C898FB32F24979E421010F90
Info (19848): Keyed hash is B82A4147FD6E88D1DD173355BDAAF7A104C8D858181983E46F742BEF6CF84653
Info: *****
Info: Running Quartus Prime Programmer
Info: Version 24.1.0 Build 115 03/21/2024 SC Pro Edition
Info: Copyright (C) 2024 Intel Corporation. All rights reserved.
Info: Your use of Intel Corporation's design tools, logic functions
Info: and other software and tools, and any partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Intel Program License
Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
Info: the Intel FPGA IP License Agreement, or other applicable license
Info: agreement, including, without limitation, that your use is for
Info: the sole purpose of programming logic devices manufactured by
Info: Intel and sold by Intel or its authorized distributors. Please
Info: refer to the Intel FPGA Software License Subscription Agreements
Info: on the Quartus Prime software download page.
Info: Processing started: Tue Jul 30 16:30:42 2024
Info: System process ID: 38864
Info: Command: quartus_pgm -m jtag -c 1 -o p:golden_top.sof@1
Info (213045): Using programming cable "Agilex 7 FPGA Starter Kit [USB-1]"
Info (213011): Using programming file golden_top.sof with checksum 0x1CC9D468 for device a5ed065bb32ae4sr0@1
Info (209060): Started Programmer operation at Tue Jul 30 16:30:49 2024
Info (18942): Configuring device index 1
Info (18943): Configuration succeeded at device index 1
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Tue Jul 30 16:30:53 2024
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1879 megabytes
Info: Processing ended: Tue Jul 30 16:30:53 2024
Info: Elapsed time: 00:00:11
Info: System process ID: 38864

```

Figure 2-23 Launch the HDMI TX demonstration from the "demo_batch" folder

Wait for a few seconds for the LCD monitor to be powered up. There will be a pre-defined video pattern shown on the monitor, as shown in **Figure 2-24**.

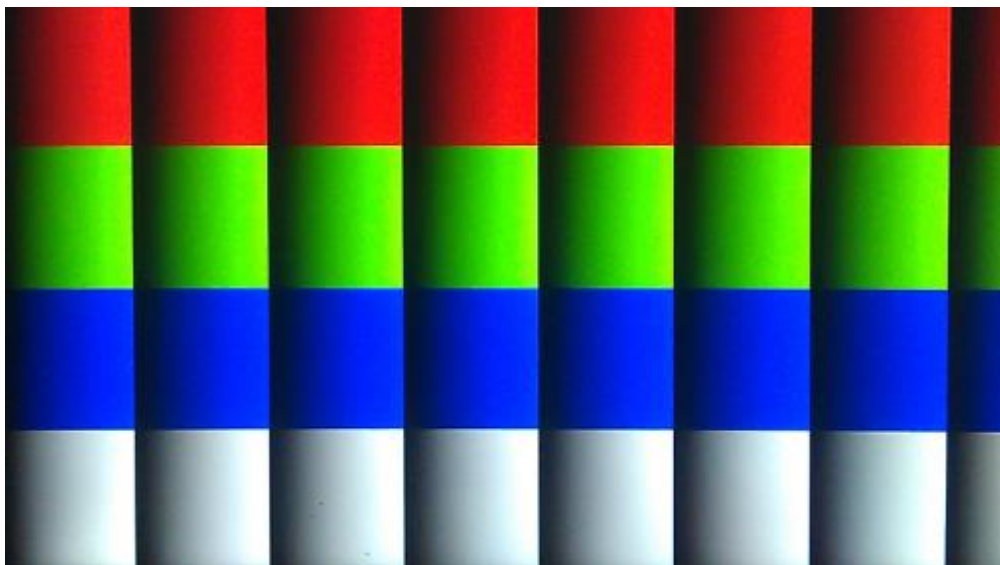


Figure 2-24 The video pattern in the HDMI TX demonstration

2.6 Low Latency 40G Ethernet Example

This 100G Ethernet example is generated according to the documents [Low Latency 40G Ethernet Intel® FPGA IP Design Example User Guide: Agilex™ 5 FPGAs and SoCs](#). The e Low Latency 40G Ethernet Intel FPGA IP is used in the example design. The IP is configured as 100GE-4 Ethernet Mode with FEC mode “IEEE 802.3 RS(528,514)(CL 91)”. This example executes the internal and external loopback test through four-channel of one QSFP28 ports on the FPGA main board. For external loopback test, a QSFP28 loopback fixture is required, otherwise only internal loopback test be available. **Figure 2-25** shows the block diagram of this demonstration.

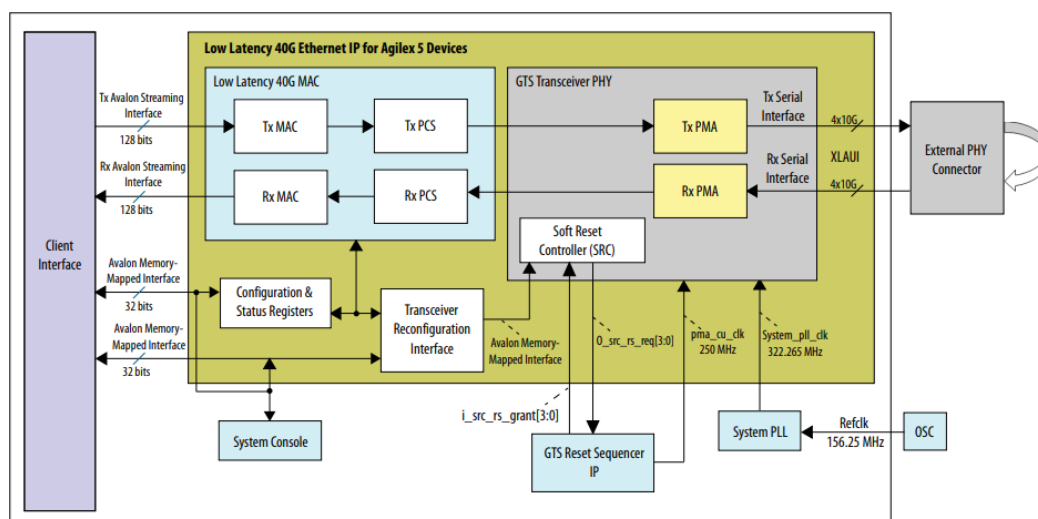


Figure 2-25 Block diagram of 100GbE demo

■ Project Information

The Quartus project is located in CD\Demonstration\FPGA folder. Project information is shown in the table below.

| Item | Description |
|------------------|--|
| Project Location | Intel_eth_e40 |
| Quartus Project | Intel_eth_e40\hardware_test_design |
| FPGA Bit Stream | Intel_eth_e40\hardware_test_design\output_files |
| Test Scrip File | Intel_eth_e40\hardware_test_design\hwtest\main.tcl |
| Quartus Version | Quartus Prime 24.3 Pro Edition |

Figure 2-26 shows the IP setup for the demonstration.

Low Latency 40G Ethernet Intel FPGA IP
intel_eth_e40

Details
Generate Example Design...

IP Analog Parameters Example Design

Main

General Options

Device family: Agilex 5

Protocol speed: 40 GbE

Ready latency: 0

PCS/PMA Options

☐ Enable SyncE

PHY reference frequency: 156.25 MHz

MAC Options

☒ Enable TX CRC insertion

☐ Enable link fault generation

☐ Enable preamble passthrough

☒ Enable MAC stats counters

☐ Enable Strict SFD check

Flow Control Options

☐ Enable MAC Flow Control

Number of queues in priority flow control: 1

Debug Options

☐ Enable JTAG to Avalon Master Bridge

Figure 2-26 40G Setup for Ethernet IP

■ Demonstration Setup

Here is the procedure to setup the demonstration. A QSFP+ loopback fixtures are required for external loopback. The **run_test_elb** enables transceiver serial loopback for external loopback.

1. Insert a QSFP+ loopback fixture into the QSFP+ port on the FPGA board, as shown in **Figure 2-27**.
2. Connect the host PC to the FPGA board using a Type-C USB cable. Please make

sure the USB-Blaster II driver is installed on the host PC.

3. Goto “**hardware_test_design/output_files**” folder and configure FPGA with the file “**eth_ex_40g.sof**”.
4. Open the **eth_ex_40g** Quartus Project and launch the System Console by selecting the menu item **Tools → System Debugging Tools → System Console** in Quartus.
5. In the System Console window, input the following commands to start the loopback test, as shown in **Figure 2-28**.

```
%cd hwtest
```

```
%source main.tcl
```

```
%run_test_elb
```

6. The loopback test report will be displayed in the Tcl Console, as shown in **Figure 2-29**.

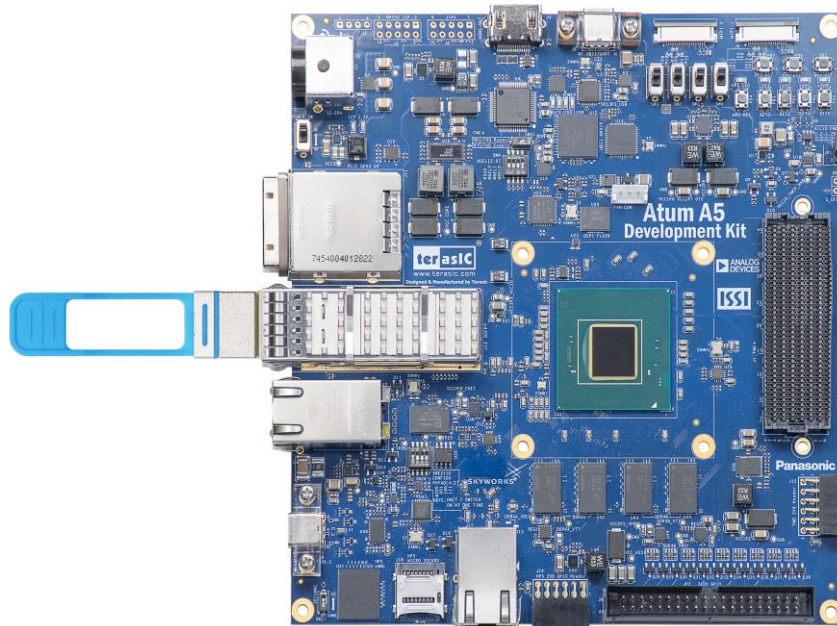


Figure 2-27 Setup QSFP+ loopback fixture

```

% cd hwtest

% source main.tcl
Available JTAG Masters:
0: /devices/A5E(C065BB32AR0|D065BB32AR0)@1#USB-1#Agilex 7 FPGA Starter Kit#192.168.1.140/(1

Type set_jtag # to select a master
Type list_jtag to display this list again
Currently selected master is 0:
/devices/A5E(C065BB32AR0|D065BB32AR0)@1#USB-1#Agilex 7 FPGA Starter Kit#192.168.1.140/(link

% run_test_elb
RX PHY Register Access: Checking Clock Frequencies (MHz)

      TXCLK      :625.0  (MHz)
      RXCLK      :624.99 (MHz)
RX PHY Status Polling

Rx Frequency Lock Status    0x0000000f

Rx Frame Error              0x00000000

Rx PHY Fully Aligned        0x00000001

Transmitting packets for 10 seconds

```

Figure 2-28 Launch the System Console for Ethernet 40G Demo

```

=====
                        STATISTICS FOR BASE 0x0900 (Rx)
=====
Fragmented Frames          : 0
Jabbered Frames            : 0
Any Size with FCS Err Frame : 0
Right Size with FCS Err Fra : 0
Multicast data Err Frames  : 0
Broadcast data Err Frames  : 0
Unicast data Err Frames    : 0
Multicast control Err Frame : 0
Broadcast control Err Frame : 0
Unicast control Err Frames : 0
Pause control Err Frames   : 0
64 Byte Frames             : 29242
65 - 127 Byte Frames       : 28266
128 - 255 Byte Frames      : 57369
256 - 511 Byte Frames      : 114714
512 - 1023 Byte Frames     : 229854
1024 - 1518 Byte Frames    : 221233
1519 - MAX Byte Frames     : 6664400
> MAX Byte Frames          : 0
Rx Frame Starts            : 0
Multicast data OK Frame    : 0
Broadcast data OK Frame    : 0
Unicast data OK Frames     : 7345078
Multicast Control Frames   : 0
Broadcast Control Frames   : 0
Unicast Control Frames     : 0
Pause Control Frames       : 0
Payload Octets OK          : 49710976181
Frame Octets OK            : 49843187585

```

Figure 2-29 Ethernet 40G loopback test report for RX

```

=====
                        STATISTICS FOR BASE 0x0800 (Tx)
=====
Fragmented Frames           : 0
Jabbered Frames             : 0
Any Size with FCS Err Frame : 0
Right Size with FCS Err Fra : 0
Multicast data Err Frames   : 0
Broadcast data Err Frames   : 0
Unicast data Err Frames     : 0
Multicast control Err Frame : 0
Broadcast control Err Frame : 0
Unicast control Err Frames  : 0
Pause control Err Frames    : 0
64 Byte Frames              : 29242
65 - 127 Byte Frames        : 28266
128 - 255 Byte Frames       : 57369
256 - 511 Byte Frames       : 114714
512 - 1023 Byte Frames      : 229854
1024 - 1518 Byte Frames     : 221233
1519 - MAX Byte Frames      : 6664400
> MAX Byte Frames           : 0
Tx Frame Starts             : 0
Multicast data OK Frame     : 0
Broadcast data OK Frame     : 0
Unicast data OK Frames      : 7345078
Multicast Control Frames    : 0
Broadcast Control Frames    : 0
Unicast Control Frames      : 0
Pause Control Frames        : 0
Payload Octets OK           : 49710976181
Frame Octets OK             : 49843187585
=====

```

Figure 2-30 Ethernet 40G loopback test report for TX

```

RX and TX packet counts match.
No Frame errors.

-----
ELB: PASS - Tx and Rx Packets Match
-----

=====
HW_INFO: Hardware Testing Result
=====
##### Hardware testing passed! #####
HW_INFO: Test Pass
HW_INFO: Test Result: [1/1] [PASS/TOTAL]

```

Figure 2-31 Ethernet 40G loopback test summary

2.7 DDR4 Test by Test Engine IP

This section explains how to test the DDR4 device on the Atum A5 board. **Figure 2-32** shows the block diagram of the example code. We begin by using the Altera **EMIF** IP to generate a DDR4 test example code. The system of this project is primarily composed of components from the **Platform Designer** in the Quartus.

In addition to the **EMIF** IP, the Altera **Test Engine** IP (traffic_generator in the block diagram) is utilized to generate test patterns for reading and writing to the DDR4 device, verifying data accuracy. The Test Engine IP is primarily operated through TCL commands. To streamline the testing process, We've created a batch file to automate the entire testing sequence.

For how to using the design example with the Test Engine IP, please refer to this document : [Using the Design Example with the Test Engine IP](#) . Please refer to the following document for further details concerning the Test Engine IP : [Test Engine FPGA IP User Guide: Agilex™ 5 and Agilex™ 7 FPGAs](#) .

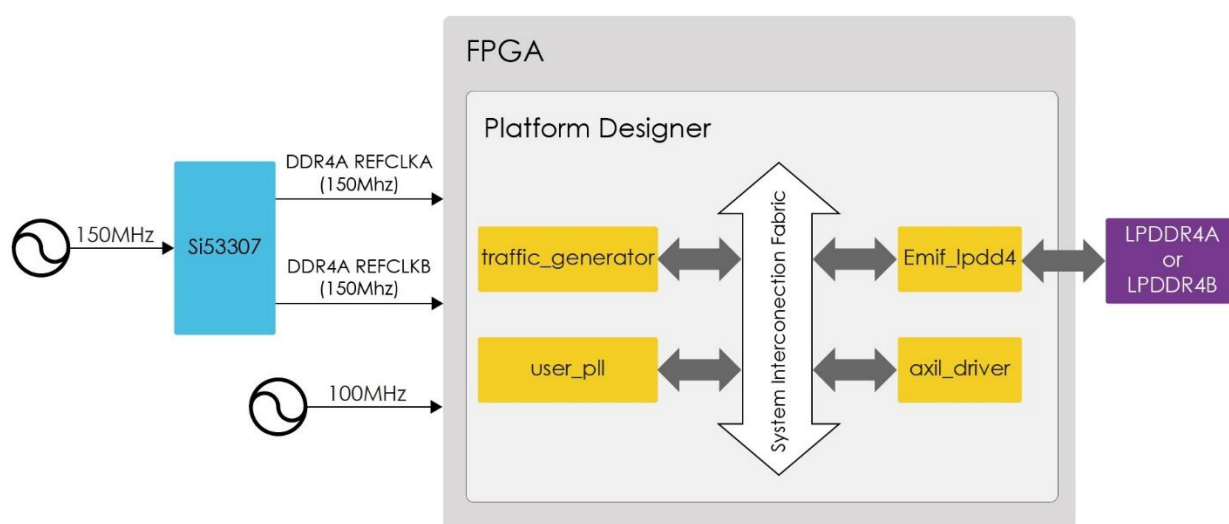


Figure 2-32 Block Diagram of the DDR4 Test

■ Design Tools

- Quartus Prime 24.3 Pro Edition or later

■ Demonstration Source Code

- Project Directory: Demonstration\FPGA\DDR4_TestEngine\
 - emif_DDR4A_TestEngine_24.3
 - emif_DDR4B_TestEngine_24.3
- Bit Stream: golden_top.sof
 - Demonstration Batch File :

- ◆ DDR4_TestEngine/emif_DDR4A_TestEngine_24.3/qii/demo_batch/
- ◆ DDR4_TestEngine/emif_DDR4B_TestEngine_24.3/qii/demo_batch/

The demo batch file includes following files:

- ◆ Batch File:
 - DDR4A.bat
 - DDR4B.bat

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Make sure Quartus Pro is installed on your PC.
2. Power on the FPGA board.
3. Use a USB Cable to connect the PC and the FPGA board and install USB Blaster II driver if necessary.
4. Execute the demo batch file “DDR4A.bat” or “DDR4B.bat” under the folder “DDR4_TestEngine/emif_DDR4A_TestEngine_24.3/qii/demo_batch/” or “DDR4_TestEngine/emif_DDR4B_TestEngine_24.3/qii/demo_batch/”.
5. After the batch file is executed successfully, a prompt message will be displayed in the command shell window.
6. The batch file will automatically load the test bit stream into the FPGA and launch the test .tcl for testing the DDR4A or DDR4B on the board.
7. TestEngine will test the DDR4 device 10 times. The test results will be displayed on the command shell (as shown in **Figure 2-33**).

```
C:\WINDOWS\system32\cmd.exe
Driver pass: 1 / 1

**** Loop 3 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 4 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 5 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 6 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 7 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 8 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1

**** Loop 9 ****
Resetting all drivers ...
Reset complete
Running traffic on all drivers ...
Traffic is now running for all drivers
Driver done: 1 / 1
Driver pass: 1 / 1
```

Figure 2-33 Testing process and results of DDR4 test

Chapter 3

Examples for HPS SoC

This chapter provides several C-code examples based on the Intel SoC Linux. These examples demonstrate major features connected to HPS interface on Agilex board such as users LED/KEY and Network Communication. All the associated files can be found in the directory CD/Demonstrations/SOC of the Agilex Kit System CD.

To install the demonstrations on the Host computer: Copy the directory Demonstrations into a local directory of your choice. ARM Toolchain is required for users to compile the c-code project.

3.1 HPS LED/KEY

This demonstration shows how to use the system call with built-in LED and GPIO driver to control the LED and KEY which are connected to HPS GPIO ports. The built-in GPIO driver is included the Agilex Kit Linux BSP.

■ How to control LED

Here is an example procedure to control the HPS LED:

1. Open LED device: Open device file “/sys/class/leds/hps_led0/brightness”.
2. Turn on/off LED: Write data to the device file for LED control. Write “1” to turn on LED, write “0” to turn off LED.
3. Close LED device: Close the device file.

■ How to Read Button Status

User space GPIO driver is used to read button status. Since linux 4.8 the GPIO sysfs interface is deprecated. User space should use the character device instead. This

library encapsulates the ioctl calls and data structures behind a straightforward API. Here is an example procedure to read the HPS Button Status:

1. Open button character device: Open character device file “/dev/gpiochip0”.
2. Configure line 1 (HPS_KEY is connected to GPIO1_IO1 in schematic) of /dev/gpiochip0 as Input GPIO
3. Read line 1 status from the button character device.
4. Close button character device: Close the character device file.

■ Function Block Diagram

Figure 3-34 shows the function block diagram of the HPS LED/KEY demonstration. LED and KEY are connected to GPIO1 Controller. The built-in GPIO driver offers access interfaces for user application program. System call open, write and close are used to control LED, and open, ioctl and closed are used to control the button.

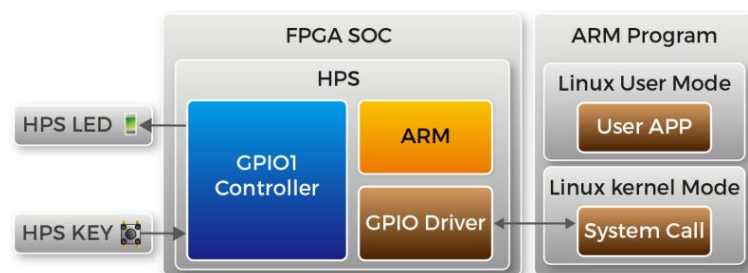


Figure 3-34 Function block diagram of HPS LED/KEY demonstration

■ Function Implement

The c project include main.c, gpio_lib.c and led_lib.c files. The main.c implements the demo main flow. The gpio_lib.c implement the KEY control functions. The led_lib.c implements LED control functions.

The led_lib implement three LED functions. With the file descriptor return by led_fd_open function, user can use led_fd_write to trun on/off the LED. Call “led_fd_write(fd_led, “1”, 2) “ will turn on the LED, and Call “led_fd_write(fd_led, “0”, 2) “ will turn off the LED. Library API is described as following:

- **int led_fd_open (unsigned int led):**

The led_fd_open function is used to open the LED device file with the specified

LED number as parameter. The function return a file descriptor for the LED device.

- **int led_fd_write (int fd, const void *buf, size_t count):**

The led_fd_write function is used to write data to the LED device file. It is used to turn on/off the LED.

- **int led_fd_close(int fd):**

The led_fd_close function is used to close a file descriptor.

The gpio_lib implement three button functions described as following:

- **GPIO_HANDLE* gpio_open_line(char *dev_name, int line, int direction):**

The gpio_open_line will open the character device file specified by *dev_name, and query the line information. All relative information are stored in a data structure GPIO_HANDLE which is dynamic created by malloc . The function returns a pointer points to a data structure GPIO_HANDLE.

- **bool gpio_get_line_value(GPIO_HANDLE *pHandle, unsigned int *pValue):**

The gpio_get_line_value reads HPS KEY status. The status is return via pValue. If HPS button is pressed, *pValue return 0, otherwise 1 is return.

- **void gpio_close_line(GPIO_HANDLE *pHandle):**

The gpio_close_line will release resource and close the open character device file.

■ Flow Control Implement

The flow control is implemented in main.c. The LED is blinking, and keep lighten when HPS KEY is pressed. The GPIO functions implemented in gpio_lib.c are used to monitor HPS KEY status. The LED functions implemented in led_lib.c is used to turn on/off the HPS LED.

Figure 3-35 shows the procedure in main.c file, you can find it's very clear.

```

line_key = gpio_open_line("/dev/gpiochip0", 1/*line 1 for KEY*/, 0 /*input*/);
↓
fd_led = led_fd_open(io_led);

loop = 20;
while (loop >= 0) {
    //gpio_get_value(io_key, &value);
    gpio_get_line_value(line_key, &key_value); // key_value is low active
    if (!key_value || bLedLight)
        led_fd_write(fd_led, "1", 2); // light led
    else
        led_fd_write(fd_led, "0", 2); // unlight led
    printf("key: %x\n", key_value);
    bLedLight = bLedLight?false:true;
    usleep(500*1000); // 0.5 second
    loop--;
}
↓
led_fd_close(fd_led);
gpio_close_line(line_key);

```

Figure 3-35 LED/KEY implemented in c code

■ Demonstration Source Code

- Build tool: ARM GNU/Linux Toolchain
- Project directory: \Demonstration\SoC\hps_led_key
- Binary file: hps_led_key
- Build command: make ('make clean' to remove all temporal files)
- Execute command: sudo ./hps_led_key

■ Demonstration Setup

1. Connect a USB cable to the Micro USB connector (J10) on the FPGA Board and the Host PC.
2. Copy the executable file "**hps_led_key**" into the microSD card under the "**/home/terasic**" folder in Linux.
3. Insert the Agilex SoC Board Linux BSP micro SD card into the Agilex SoC Board.
4. Power on the Agilex SoC Board.
5. Launch Putty to establish the connection between the UART port of Agilex SoC Board and the Host PC.
6. In the Putty UART terminal, type user name "terasic" and password "123" to login Linux.
7. Copy the executable file "hps_led_key" into the "/home/terasic" folder in Linux.
8. Type "sudo ./hps_led_key" in the UART terminal to start the program. Input password "123" if system query password for terasic.
9. You will see the key status is shown in Putty UART terminal as shown in **Figure 3-36**.
10. Press HPS KEY will make key value become 0 and HPS LED keep lighten.

11. The program will be automatically terminated in 20 seconds, or press CTRL+C to terminate the program immediately.

```
terasic@localhost:~$ sudo ./hps_led_key  
/sys/class/leds/hps_led0/brightness  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 1  
key: 0  
key: 1  
key: 1  
^C  
terasic@localhost:~$
```

Figure 3-36 LED/KEY test

3.2 Network Socket

This demonstration shows how two remote application processes communication via socket in client-server model. Based on this design example, developers can make their Linux Application Software, run on SoC FPGA boards and easily communicate with other Hosts via a network socket.

■ Sockets

Sockets are the fundamental technology for programming software to communicate on the transport layer of networks shown in [Figure 3-37](#). A socket provides a bidirectional communication endpoint for sending and receiving data with another socket. Socket connections normally run between two different computers on a LAN, or across the Internet, but they can also be used for interposes communication on a single computer.

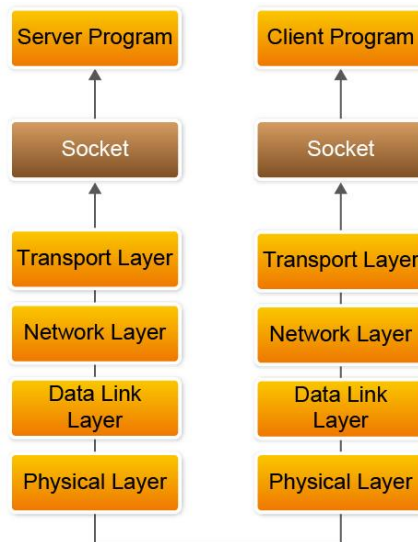


Figure 3-37 Communicate on a network via a socket

■ Client Server Model

Most intercrosses' communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server typically to makes a request for information. A good analogy is a person who makes a phone call to another person.

Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established.

Notice also that once a connection is established, both sides can send and receive information.

The system calls for establishing a connection which is somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of an intercross's communication channel. The two processes each establish their own socket. **Figure 3-38** shows the communication diagram between the client and server.

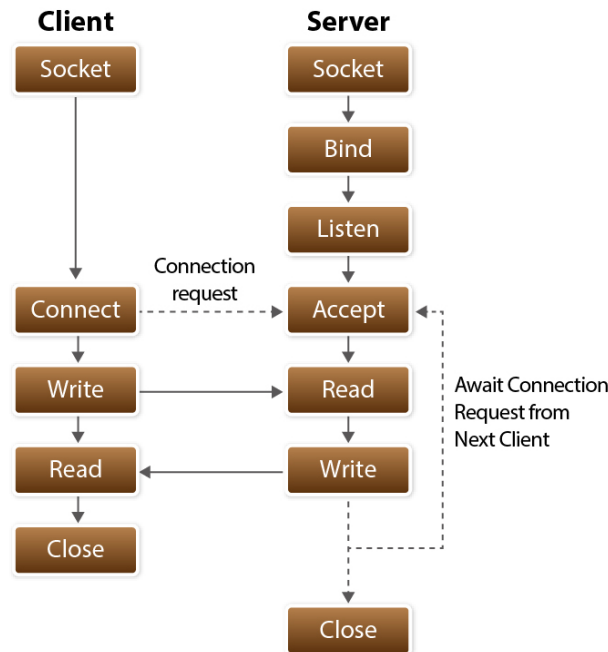


Figure 3-38 Client and Server communication

The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the **socket()** system call
- Connect the socket to the address of the server using the **connect()** system call
- Send and receive data. There are a number of ways to do this, but the simplest is to use the **read()** and **write()** system calls.

The steps involved in establishing a socket on the **server** side are as follows:

- Create a socket with the **socket()** system call
- Bind the socket to an address using the **bind()** system call. For a server socket on the Internet, an address consists of a port number on the Host machine.
- Listen for connections with the **listen()** system call
- Accept a connection with the **accept()** system call. This call typically blocks until a client connects with the server.
- Send and receive data. There are a number of ways to do this, but the simplest is to use the **read()** and **write()** system calls.

■ Example Code Explanation

The example design contains two projects. One is socket server project, and one is socket client project. The SOCK_STREAM socket type is used in the design. The Linux Socket Library is used to provide socket functions, so remember to include the socket

The major function of socket server program is to create a socket server based on the given port number and waiting a client to request to establish a connection. When a connection is established, the server is waiting for an incoming text message. When a message is received, it will show the receiver message on the console terminal, then send the message “I got your message” to the client socket, and then close the server program. **Figure 3-39** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK_STREAM socket, **bind** API is used to bind the socket to any incoming address and a specified port number. For connection, **listen** API is used to make the socket as a passive socket that is, as a socket that will be used to accept the incoming connection, and **accept** API is used to accept the incoming connection. The **accept** blocks until a client connects with the server. Data receiving and sending is implemented by the **read** and **write** API, and **close** is used to close the socket.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);  
if (sockfd < 0)   
    error("ERROR opening socket");  
bzero((char *) &serv_addr, sizeof(serv_addr));  
portno = atoi(argv[1]);  
serv_addr.sin_family = AF_INET;  
serv_addr.sin_addr.s_addr = INADDR_ANY;  
serv_addr.sin_port = htons(portno);  
if (bind(sockfd, (struct sockaddr *) &serv_addr,  
    sizeof(serv_addr)) < 0)   
    error("ERROR on binding");  
listen(sockfd,5);  
clilen = sizeof(cli_addr);  
newsockfd = accept(sockfd,   
    (struct sockaddr *) &cli_addr,   
    &clilen);  
if (newsockfd < 0)   
    error("ERROR on accept");  
bzero(buffer,256);  
n = read(newsockfd,buffer,255);  
if (n < 0) error("ERROR reading from socket");  
printf("Here is the message: %s\n",buffer);  
n = write(newsockfd,"I got your message",18);  
if (n < 0) error("ERROR writing to socket");  
close(newsockfd);  
close(sockfd);
```

Figure 3-39 Socket Server Code

The major function of the socket client program is to create a connection based on given Hostname (or IP address) and Host port. When a connection is established, it will show “Please enter the message:” message on console terminal to ask users to input a

message. After get user's input message, the message is sent to a remote socket server via the socket. If the remote server socket received the message, it will return a message "I got the message". The client program will show the received message on the console terminal and exit the program. **Figure 3-40** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK_STREAM socket, **connect** API is used to connect the remove socket sever based on the given Hostname (or IPv4 Address) and port number. Data receiving and sending is implemented by **read** and **write** API, and **close** is used to **close** the socket.

```

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr, "ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr,
      server->h_length);
serv_addr.sin_port = htons(portno);
if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
    error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
    error("ERROR reading from socket");
printf("%s\n",buffer);
close(sockfd);

```

Figure 3-40 Socket Client Code

■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-41**. The Demonstration folder contains three platform subfolders: **arm**, **linux** and **windows**. The project under the **arm** folder is designed for SoC FPGA board. The project under **linux** folder is designed for Linux running on Linux PC. The project under **windows** folder is designed for SoC EDS Shell running on Windows PC. Each platform subfolder contains socket_client and socket_server project folders.



Figure 3-41 Source Code Folder Tree

The `socket_client` project includes a Makefile and a source file `main.c`. For different platforms, the Makefile content is different, but the `main.c` content is the same. The `socket_server` project has the file project architecture.

■ Demonstration Setup

Here we show the procedure to execute the socket client-server communication demonstration. In this setup procedure, the server program is running on Intel SoC FPGA board and the Socket Client is running on Windows PC.

1. Connect the Agilex SoC Board to Network via Ethernet port (J9).
2. Connect a USB cable to the Micro USB connector (J10) on the board and the Host Windows PC.
3. Copy the executable file "**socket_server**" into the microSD card under the `"/home/terasic"` folder in Linux. (board Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Agilex SoC Board Linux BSP micro SD card into the Agilex SoC Board.
5. Power on the Agilex SoC Board.
6. Launch the Putty to connect Agilex SoC Board via the USB-to-UART link.
7. In the Putty, type user name "**terasic**" and password "**123**" to login Linux.
8. Type "**ifconfig**" to query the IP address which will be used in `socket_client`.
9. Type "**./socket_server 2020**" to launch the server program with port number 2020 as shown in **Figure 3-42**. The port number can be any value between 2000 and 63500.

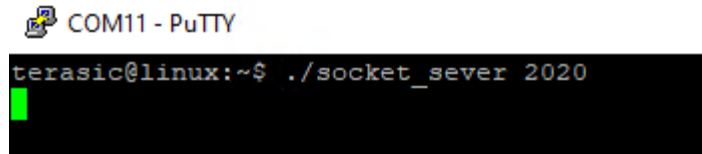


Figure 3-42 Start Socket Server

Here is the procedure to start the socket client program and communicate with the client server program:

1. Make sure the WSL is installed on your Windows and the Windows is connected to a network.
2. Launch WSL.
3. Copy the client program (linux/socket_client/socket_client) in the example kit to the WSL.
4. In the WSL, change the current directory to the directory where socket_client is located.
5. Then, type “./socket_client <ip address> 2020” to launch the client program to connect to the Host server with port number 2020 as shown in **Figure 3-43**.

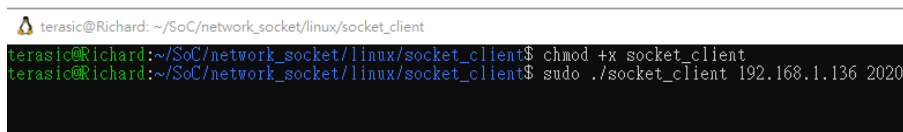


Figure 3-43 Start Client Program

6. If connection is established successfully, a prompt message “Please enter the message.” will appear. Type “**hello**”, then an echo message “**I got your message**” will be sent from the client server and shown on terminal as shown in **Figure 3-44**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-45**.

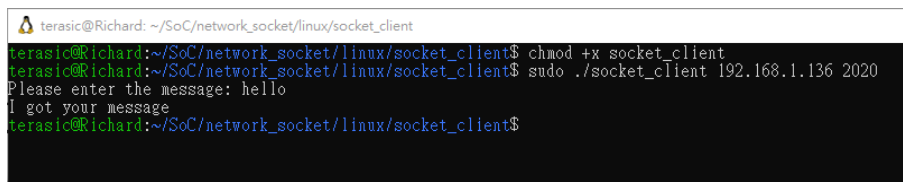
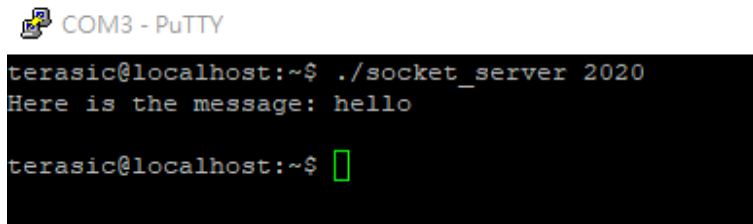


Figure 3-44 Send Message in Client Program



```
COM3 - PuTTY
terasic@localhost:~$ ./socket_server 2020
Here is the message: hello
terasic@localhost:~$
```

Figure 3-45 Server dumps received message

3.3 Build C/C++ Project

This section describes how to recompile the above C/C++ project included in the System CD.

First, user need to download and install ARM GNU/Linux tool chain:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. Type
“wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz”
4. Type “tar xf gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz”
5. Type “export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin:\$PATH”
6. Type “export CROSS_COMPILE=aarch64-none-linux-gnu-”
7. Type “git clone <https://github.com/altera-opensource/intel-socfpga-hwlib>” to download HPS hardware library.

Here is the procedure to compile the example projects in System CD:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. Type “export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin:\$PATH”
4. Type “export CROSS_COMPILE=aarch64-none-linux-gnu-”
5. Copy the CD Demo project into the Linux System and go to the project folder.
6. Type “make” to build project as shown in **Figure 3-46**.


```

terasic@Richard:~/SoC/hps_led_key$ make clean
rm -rf hps_led_key main.o led_lib.o gpio_lib.o *.objdump *.map
terasic@Richard:~/SoC/hps_led_key$ make
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c main.c -o main.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c led_lib.c -o led_lib.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c gpio_lib.c -o gpio_lib.o
aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall main.o led_lib.o gpio_lib.o -o hps_led_key
aarch64-none-linux-gnu-nm hps_led_key > hps_led_key.map
terasic@Richard:~/SoC/hps_led_key$ ls
Makefile gpio_lib.c gpio_lib.h gpio_lib.o hps_led_key hps_led_key.map led_lib.c led_lib.h led_lib.o main.c main.o
terasic@Richard:~/SoC/hps_led_key$

```

Figure 3-46 Build C/C++ Project

Chapter 4

Transceiver Verification

This chapter describes how to quickly verify the FPGA transceivers via the QSFP+ connector.

4.1 Transceiver Test Code

The transceiver test code is used to verify the transceiver channels for the QSPF+ ports through an external loopback method. The transceiver channels are verified with the data rates 10.3125 Gbps with PRBS31 test pattern

4.2 Loopback Fixture

To enable an external loopback of the transceiver channels, QSFP+ loopback fixtures, as shown in **Figure 4-1**, are required. The manuafature of this loopback fixtures is “Molex” and “74763-0025” is the part number.



Figure 4-1 QSFP+ Loopback Cable

Figure 4-2 shows the FPGA board with four QSFP+ loopback fixtures installed.

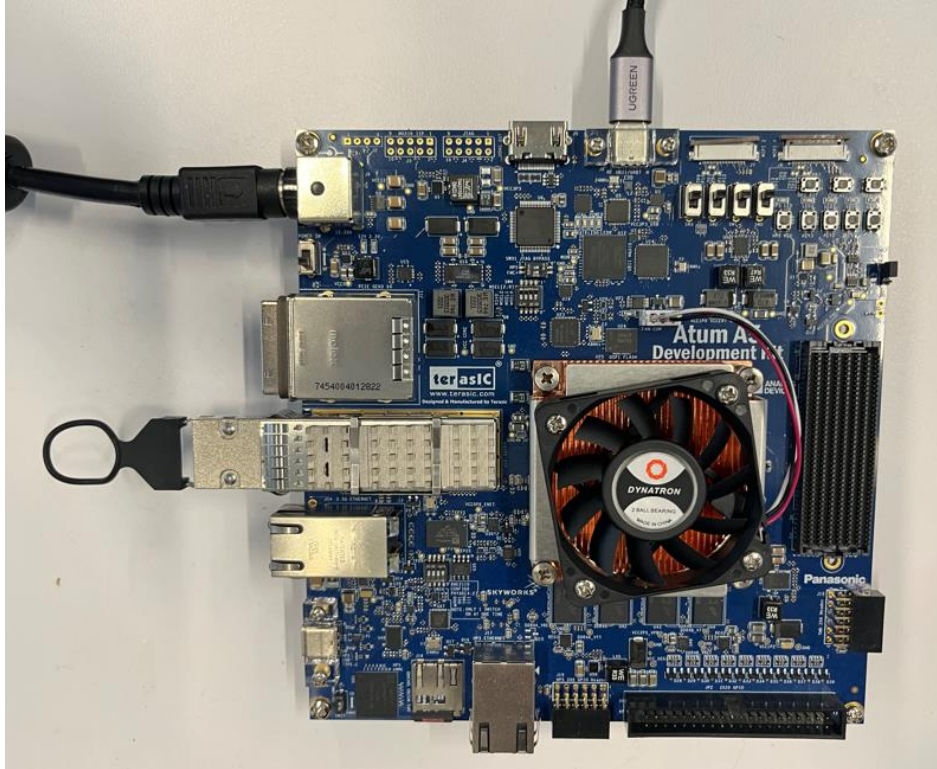


Figure 4-2 QSFP+ Loopback installed in the board

4.3 Testing by Transceiver Test Code

The transceiver test code is available in the folder System CD\Tool\Transceiver_Test.

Figure 4-3, **Figure 4-4** and **Figure 4-5** shows the GTS PMA/FEC Direct PHY settings in the test code. The data rate of each transceiver channel is set to 10312.5 Mbps and the PMA modulation type is NRZ. So the 40Gbps QSPF+ loopback test code is implemented (4 channels in total). Also, the GTS Reference and System PLL setting is shown in **Figure 4-6**.

Design Environment

This component supports multiple interface views:

System

Common Datapath Options

PMA configuration rules:
Basic

Number of PMA lanes:
4

Datapath clocking mode:
System PLL

System PLL frequency:
322.265625
MHz

PMA mode:
Duplex

PMA data rate:
10312.5
Mbps

PMA parallel clock frequency:
322.265625
MHz

PMA width:
32

☐ Provide separate interface for each PMA

☐ Enable refclock to core

Figure 4-3 The Transceiver PHY setting

| | | |
|-------------------------------------|----------------------------------|-----|
| RX PMA | | |
| <input type="checkbox"/> | Enable rx_cdr_divclk | |
| RX CDR Settings | | |
| Output frequency: | 5156.250000 | MHz |
| VCO frequency: | 10312.500000 | MHz |
| RX CDR reference clock frequency: | 156.250000 | MHz |
| CDR lock mode: | auto | |
| <input type="checkbox"/> | Enable rx_set_locktoref port | |
| RX PMA Interface | | |
| RX PMA interface FIFO mode: | Elastic | |
| <input type="checkbox"/> | Enable rx_pmaif_fifo_empty port | |
| <input type="checkbox"/> | Enable rx_pmaif_fifo_pempty port | |
| <input type="checkbox"/> | Enable rx_pmaif_fifo_pfull port | |
| RX Core Interface | | |
| RX Core Interface FIFO | | |
| RX core interface FIFO mode: | Phase compensation | |
| <input checked="" type="checkbox"/> | Enable RX double width transfer | |
| <input type="checkbox"/> | Enable rx_fifo_full port | |
| <input type="checkbox"/> | Enable rx_fifo_empty port | |
| <input type="checkbox"/> | Enable rx_fifo_pfull port | |
| <input type="checkbox"/> | Enable rx_fifo_pempty port | |
| <input type="checkbox"/> | Enable rx_fifo_rd_en port | |
| RX Clock Options | | |
| Selected rx_clkout clock source: | sys PLL clock | |
| rx_clkout clock div by: | 2 | |
| Frequency of rx_clkout: | 161.132812 | MHz |
| <input type="checkbox"/> | Enable rx_clkout2 port | |
| RX User Clock Setting | | |
| RX user clock div by: | 100 | |

Figure 4-4 The Transceiver PHY setting

| TX Datapath Options | RX Datapath Options | FEC Options | PCS Options | Avalon Memory-Mapped Interface |
|--|---------------------|-------------|-------------|--------------------------------|
| TX PMA | | | | |
| TX PLL Settings | | | | |
| Output frequency: | 5156.250000 | | | MHz |
| VCO frequency: | 10312.500000 | | | MHz |
| <input type="checkbox"/> Enable TX PLL cascade mode | | | | |
| <input type="checkbox"/> Enable TX PLL fractional mode | | | | |
| TX PLL integer mode reference clock frequency: | 156.250000 | | | MHz |
| TX PLL fractional mode reference clock frequency: | 156.250000 | | | MHz |
| TX PMA Interface | | | | |
| TX PMA interface FIFO mode: | Elastic | | | |
| <input type="checkbox"/> Enable tx_pmaif_fifo_empty port | | | | |
| <input type="checkbox"/> Enable tx_pmaif_fifo_pempty port | | | | |
| <input type="checkbox"/> Enable tx_pmaif_fifo_pfull port | | | | |
| TX Core Interface | | | | |
| TX Core Interface FIFO | | | | |
| <input type="checkbox"/> Enable custom cadence generation ports and logic | | | | |
| <input checked="" type="checkbox"/> Enable tx_cadence_slow_clk_locked port | | | | |
| TX core Interface FIFO Mode: | Phase compensation | | | |
| <input checked="" type="checkbox"/> Enable TX double width transfer | | | | |
| <input type="checkbox"/> Enable tx_fifo_full port | | | | |
| <input type="checkbox"/> Enable tx_fifo_empty port | | | | |
| <input type="checkbox"/> Enable tx_fifo_pfull port | | | | |
| <input type="checkbox"/> Enable tx_fifo_pempty port | | | | |
| TX Clock Options | | | | |
| Selected tx_clkout clock source: | Sys PLL Clock | | | |
| tx_clkout clock div by: | 2 | | | |
| Frequency of tx_clkout: | 161.132812 | | | MHz |
| <input type="checkbox"/> Enable tx_clkout2 port | | | | |
| TX User Clock Setting | | | | |
| TX user clock div by: | 100 | | | |
| TX user clock Frequency: | 103.125 | | | MHz |

Figure 4-5 The Transceiver PHY setting

Figure 4-6 The GTS Sytem PLL Clock setting

The FPGA transceiver PMA setting used are shown in the table below.

| Direction | Item | Value |
|-----------|--------------|-------|
| TX | pre_tap_2 | 0 |
| | pre_tap_1 | 0 |
| | main_tap | 45 |
| | post_tap_1 | 4 |
| RX | Auto Default | |

Here are the procedures to perform transceiver channel test:

1. Copy the Transceiver_Test folder to your local disk.
2. Ensure that the FPGA board is NOT powered on.
3. Plug-in the QSPF+ loopback fixtures.
4. Connect your FPGA board to your PC with a USB cable.
5. Power on the FPGA board
6. Execute "test.bat" in the Transceiver_Test folder under your local disk.

7. The batch file will download .sof and .elf files, and start the test immediately. The test result is shown in the command shell terminal, as shown in **Figure 4-7**. Note, the result show “error count = 0” means the test is pass.
8. To terminate the test, press one of the BUTTON0~1 buttons on the FPGA board.

```
E:\intelFPGA_pro\22.3\quartus\bin64\nios2-terminal.exe
Info: Processing ended: Tue May 16 16:44:06 2023
Info: Elapsed time: 00:00:27
Info: System process ID: 2444
Using cable "Agilex 7 FPGA Starter Kit [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 151KB in 0.2s (755.0KB/s)
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x00540238
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Agilex 7 FPGA Starter Kit [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Transceiver for QSFP28 testing...
Press buttons on the board can terminate the testing.
Init...
===== Time Elapsed (d h:m:s): 0 0:0:0 =====
QSFP28 xcvr 25.78125Gpbs x4, Numbers of bit tested: (0.000E+00) x4
CH-0 error count:0
CH-1 error count:0
CH-2 error count:0
CH-3 error count:0

===== Time Elapsed (d h:m:s): 0 0:0:5 =====
QSFP28 xcvr 25.78125Gpbs x4, Numbers of bit tested: (1.289E+11) x4
CH-0 error count:0
CH-1 error count:0
CH-2 error count:0
CH-3 error count:0

===== Time Elapsed (d h:m:s): 0 0:0:10 =====
QSFP28 xcvr 25.78125Gpbs x4, Numbers of bit tested: (2.578E+11) x4
CH-0 error count:0
CH-1 error count:0
CH-2 error count:0
CH-3 error count:0

===== Time Elapsed (d h:m:s): 0 0:0:15 =====
QSFP28 xcvr 25.78125Gpbs x4, Numbers of bit tested: (3.867E+11) x4
CH-0 error count:0
CH-1 error count:0
CH-2 error count:0
CH-3 error count:0
```

Figure 4-7 QSFP+ Transceiver Loopback Test in Progress

Chapter 5

Additional Information

5.1 Getting Help

Here are the addresses where you can get help if you encounter problems:

■ Terasic Technologies

No.80, Fenggong Rd., Hukou Township, Hsinchu County 303035. Taiwan

Email: support@terasic.com

Web: www.terasic.com

Atum A5 Development Kit Web: ATUM A5.terasic.com

■ Revision History

| Date | Version | Changes |
|---------|-------------------|--|
| 2024.07 | First publication | |
| 2024.10 | V1.1 | Add section for Si5391B IP and Clock Controller demo for Si5391B |
| 2025.01 | V1.2 | Add section 2.7 DDR4 Test And Add Chapter 4 Transceiver Verification |
| | | |
| | | |
| | | |
| | | |

