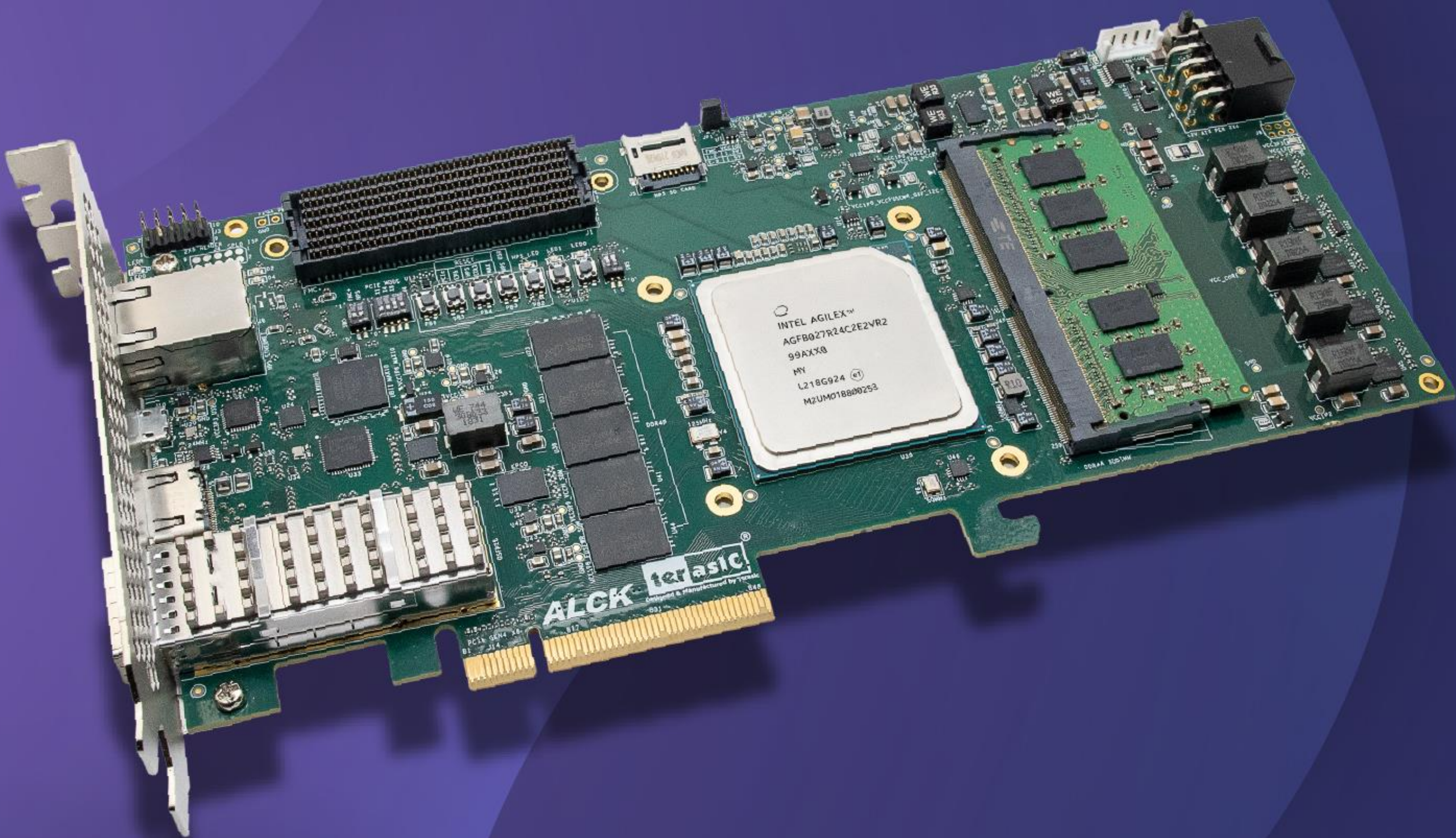
Agilex 7 FPGA Starter Kit

Demonstration Manual



www.terasic.com

Contents

Chapter 1

Overview

This Manual will introduce the various application demonstrations on **Agilex 7 FPGA Starter Kit(A7SK)**. 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)**
- **PCI Express Refernce Desing for Linux (Chpater 4)**
- **PCI Express Refernce Desing for Windows (Chpater 5)**
- **Transceiver Verification (Chpater 6)**

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

- **Intel Quartus® Prime Pro Edition Software Version 23.2 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 **Agilex 7 FPGA Starter Kit(A7SK)**. 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 A7SK System CD.

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

This demonstration shows how to use the Nios II 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.

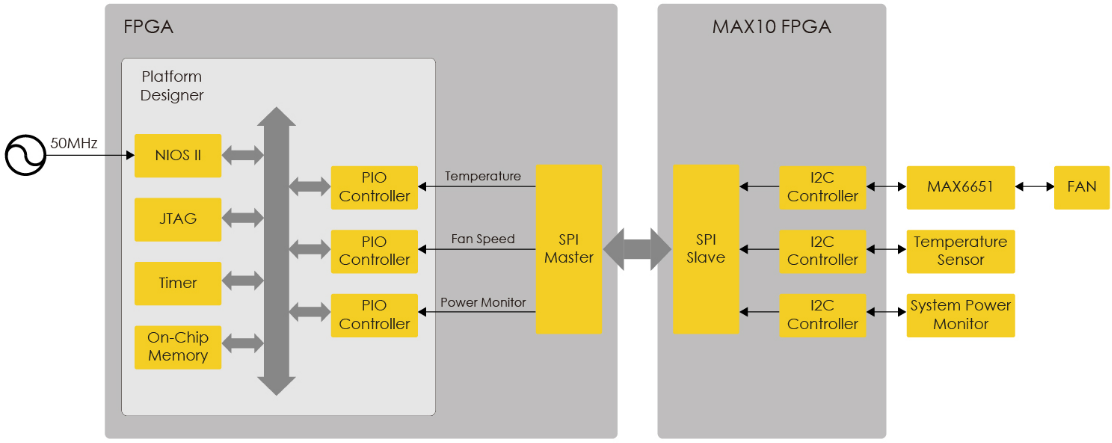


Figure 2-1 Block Diagram of the Nios II Basic Demonstration

The system provides a menu in nios-terminal, 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.

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. A power monitor IC (LTC2945) embedded on the board can monitor real-time current and power. This IC can work out current/power value as multiplier and divider are embedded in it. There is a sense resistor R84 (0.003 Ω) for LTC2945 in the circuit, when power on the Board, there will be a voltage drop (named ∆SENSE Voltage) on R5. Based on sense resistors, the program of power monitor can calculate the associated voltage, current and power consumption.

■ 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, test.sh

■ 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 II program is downloaded and executed successfully, a prompt message will be displayed in nios2-terminal.
6. For temperature, power monitor and fan test, please input key ‘0’ and press ‘Enter’ in the nios-terminal, as shown in **Figure 2-3**.

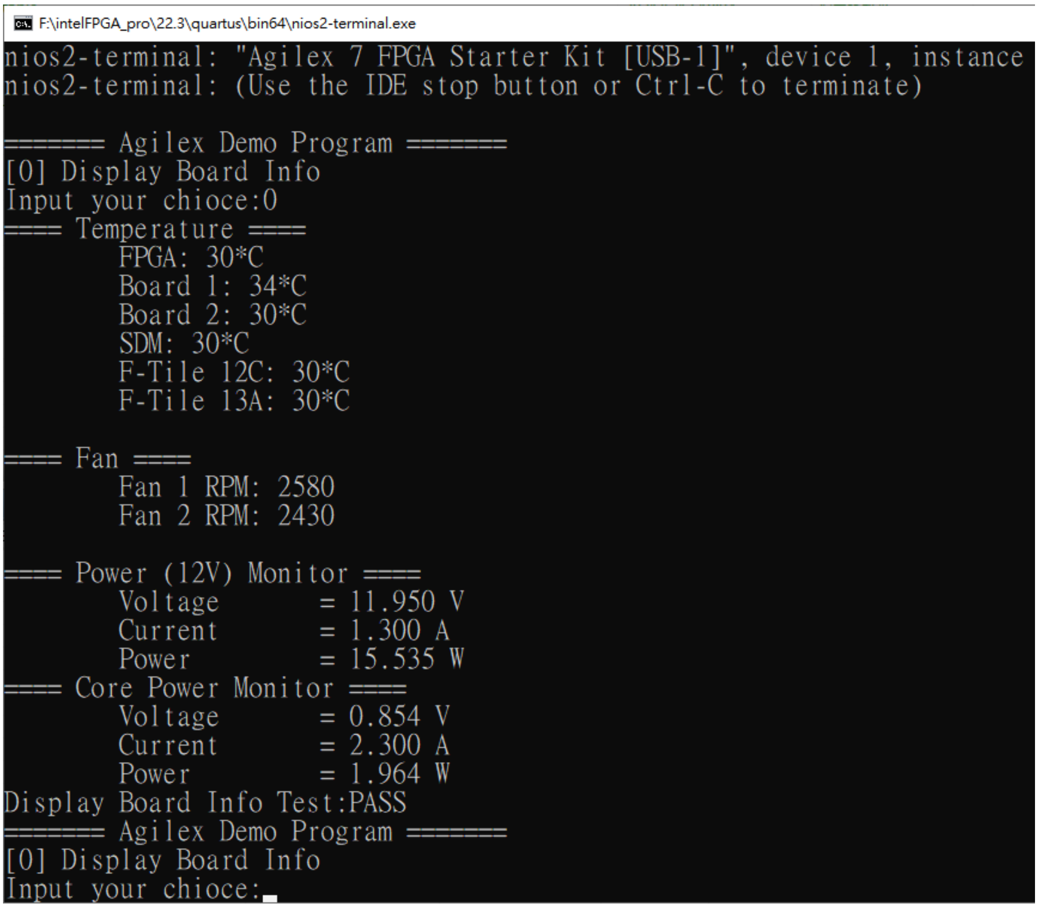


Figure 2-3 Board Info Demo

2.2 DDR4 SDRAM RTL Test

This demonstration performs a memory test function using RTL code on the DDR4 SO-DIMM and on-board DDR4 SDRAM on the board. Both DDR4 devices are controlled by FPGA fabric. The memory size of each DDR4 SDRAM used in this test is 8 GB.

■ Function Block Diagram

Figure 2-4 shows the function block diagram of this demonstration. There are two DDR4 SDRAM controllers (DDR4A and DDR4B) in this project. All of the controllers use 33.333 MHz as a reference clock. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-1.** The test program will write data into SDRAM, after writing 16GB capacity, it will read the values from SDRAM and check whether there is any error.

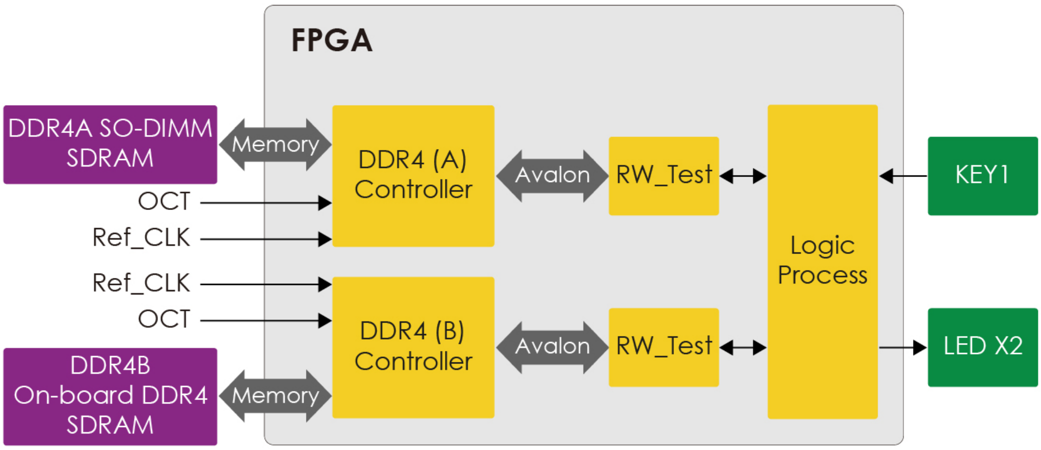


Figure 2-4 Block diagram of the DDR4 RTL demonstration

Table 2-1 DDR4 clock frequency for each speed grade of FPGA

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
AGFB014R24B2E2V	1333 (DDR4 2666)

■ Agilex External Memory Interfaces

To use Agilex External Memory Interfaces controller for DDR4 SO-DIMM SDRAM(DDR4A) and on-board DDR4 SDRAM(DDR4B), please perform the two major steps below:

1. Create correct pin assignments for the DDR4 SODIMM and on-board DDR4 SDRAM.
2. Setup correct parameters in the dialog of the **Agilex FPGA External Memory Interfaces**.

■ Design Tools

- Quartus Prime 23.0 Pro Edition or later

■ Demonstration Source Code

- Project Directory: Demonstration\FPGA\RTL_DDR4_Test
- Bit Stream: ALCK.sof
- Demonstration Batch File : RTL_DDR4_Test\demo_batch

  The demo batch file includes following files:

- Batch File: test.bat
- FPGA Configuration File: ALCK.sof

Demonstration Setup

1. Make sure Quartus Prime Pro Edition is installed on the Host PC.
2. Connect the board to the Host PC via the USB cable. Install the USB-Blaster II driver if necessary.
3. Power on the board.
4. Execute the demo batch file “test.bat” under the batch file folder \RTL_DDR4_Test\demo_batch.
5. Press **Button1** (see **Figure 2-5**) to start DDR4 write & loopback verify process. It will take about 1 second to perform the test. While testing, the LED will blink. When LED stop blinking it means the test process is done.
6. The test result will show on LED0 and LED1. The **LED0** represents the test result for the DDR4A (DDR4 SO-DIMM socket), the **LED1** represents the test result for the DDR4B(on-board DDR4 SDRAM). if the LED0/LED1 light on, it means the test result is passed. If the LED0/LED1 is off, it means the test result is failed.

7. User can press **Button1** again to regenerate the test control signals for a repeat test.

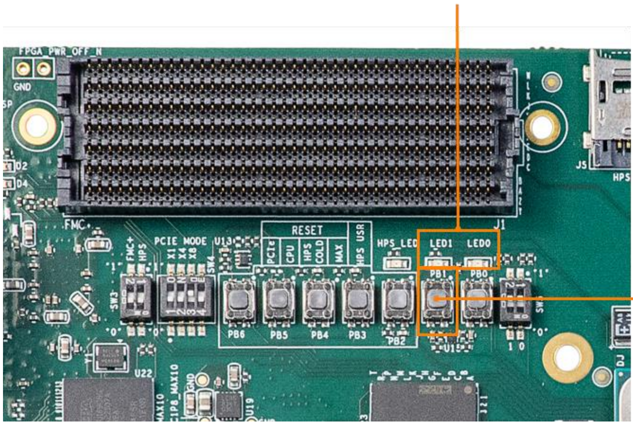


Figure 2-5 Location of the Button and LED on the board

2.3 DDR4 SDRAM Test by Nios II

Many applications use a high performance RAM, such as a DDR4 SDRAM, to provide temporary storage. In this demonstration hardware and software designs are provided to illustrate how to perform DDR4 memory access in the Platform Designer (formerly Qsys). We describe how the memory controller Agilex External Memory Interfaces is used to access the two DDR4 SO-DIMM SDRAM socket and on-board DDR4 SDRAM on the FPGA board, and how the Nios II processor is used to read and write the SDRAM for hardware verification. The DDR4 SDRAM controller handles the complex aspects of using the DDR4 SDRAM by initializing the memory devices, managing the SDRAM banks, and keeping the devices refreshed at the appropriate intervals.

■ System Block Diagram

Figure 2-6 shows the system block diagram of this demonstration. In the Platform Designer (formerly Qsys), one 50 MHz, 33.33 Mhz OSC and clock buffer(Si53307) are used. The OSC and clock buffer will provide two 33.333Mhz clock to the DDR4 SO-DIMM socket (DDR4A) and on-board DDR4 SDRAM(DDR4B) as the reference clock. There are two DDR4 Controllers which are used in the demonstrations. Each controller is responsible for one DDR4 SO-DIMM socket and on-board DDR4 SDRAM (DDR4A and DDR4B). Each DDR4 controllers are configured as 8GB DDR4 controller. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-2**. The Nios II processor is used to perform the memory test. The Nios II program is running in the On-Chip Memory. A PIO Controller is used to monitor buttons status which is used to trigger starting memory testing.

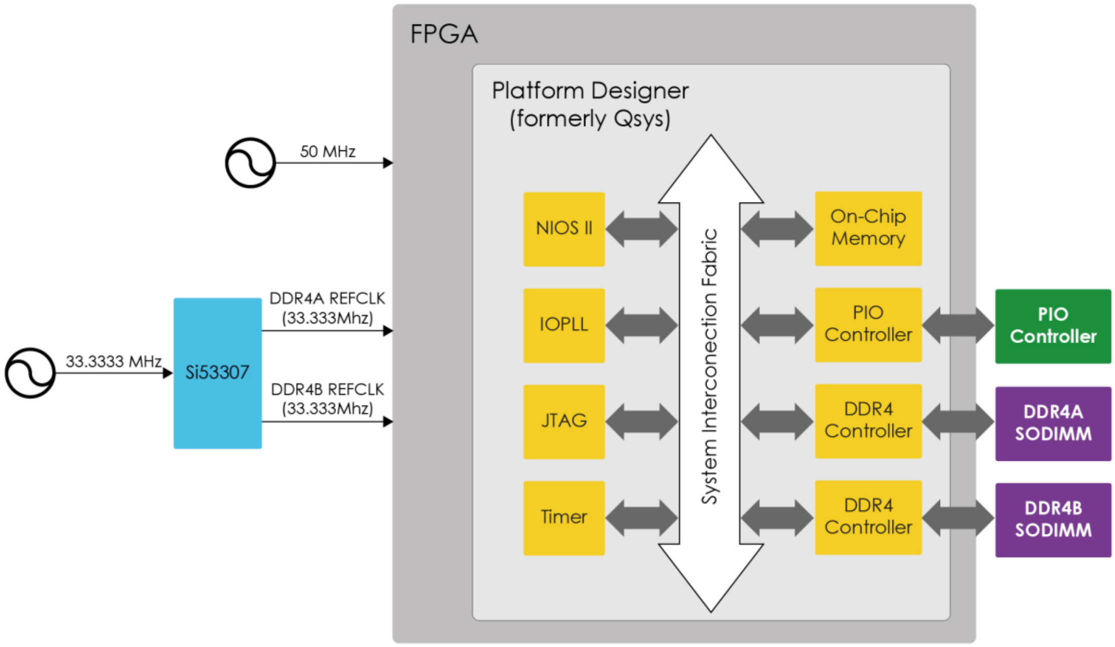


Figure 2-6 Block diagram of the DDR4 Basic Demonstration

The system flow is controlled by a Nios II program. First, the Nios II program writes test patterns into the whole 8GB of SDRAM. Then, it calls Nios II system function, alt_dache_flush_all(), to make sure all data has been written to SDRAM. Finally, it reads data from SDRAM for data verification. Maybe the process takes a long time, and there is a quick test. The Nios II program writes a constant pattern into the address line and data line and reads it back for verification. The program will show progress in Nios II terminal when writing/reading data to/from the SDRAM. When verification process is completed, the result is displayed in the Nios II terminal.

Table 2-2 DDR4 clock frequency for each speed grade of FPGA

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
AGFB014R24B2E2V	1333 (DDR4 2666)

■ Design Tools

- Quartus Prime 23.0 Pro Edition

■ Demonstration Source Code

- Quartus Project directory: NIOS_DDR4_Test
- Nios II Eclipse: NIOS_DDR4_Test \software

■ Nios II Project Compilation

Before you attempt to compile the reference design under Nios II Eclipse, make sure the project is cleaned first by clicking ‘Clean’ from the ‘Project’ menu of Nios II Eclipse.

■ Demonstration Batch File

Demo Batch File Folder: NIOS_DDR4_Test\demo_batch

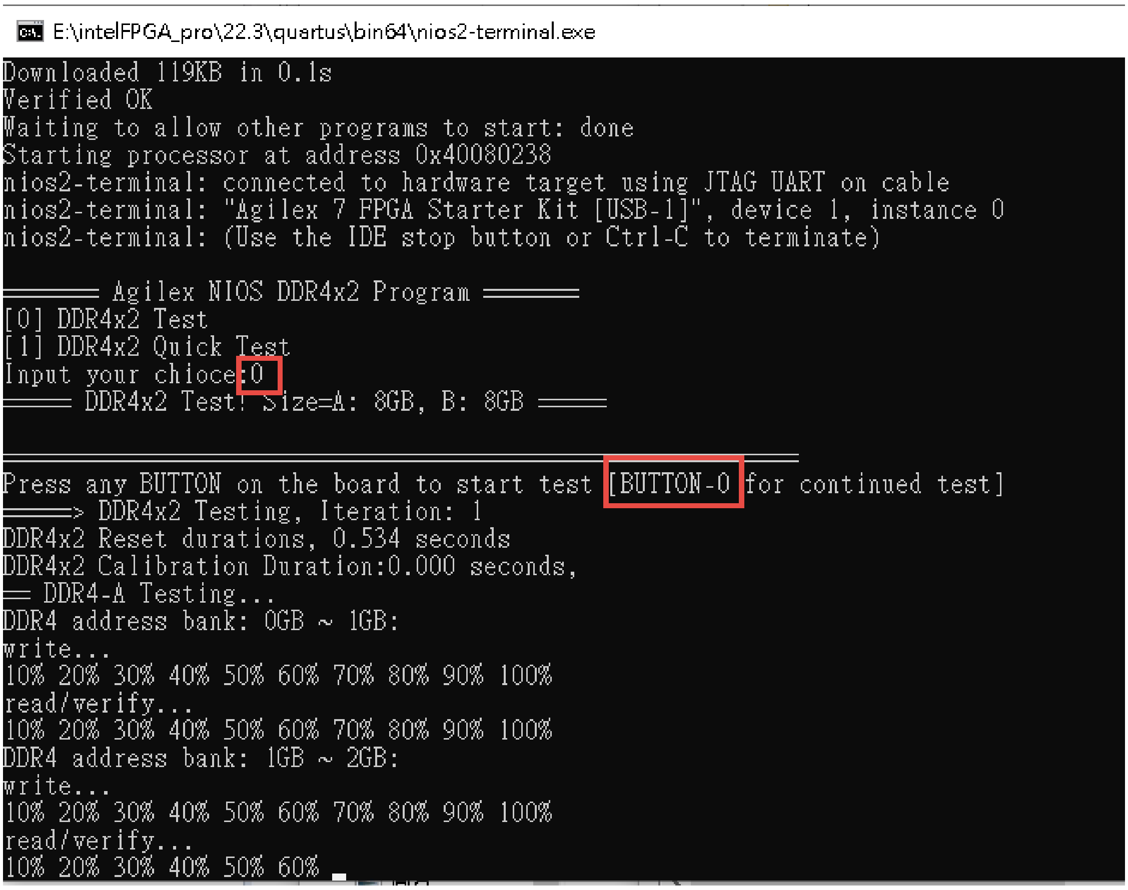
The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat, test.sh
- FPGA Configure File: ALCK.sof
- Nios II Program: MEM_Test.elf

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Make sure Quartus Prime and Nios II are 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 “test.bat” under the folder “NIOS_DDR4_Test\demo_batch”.
5. After the Nios II program is downloaded and executed successfully, a prompt message will be displayed in the nios2-terminal.
6. For DDR4 test, please input key ‘0’ and press ‘Enter’ in the nios2-terminal as shown in **Figure 2-7**. The program will display progressing and result information.
7. For DDR4 quick test, please input key ‘1’ and press ‘Enter’ in the nios2-terminal as shown in **Figure 2-8**. The program will display progressing and result information. Press Button0~Button1 of the FPGA board to start SDRAM verify process, and press Button0 for continued test.

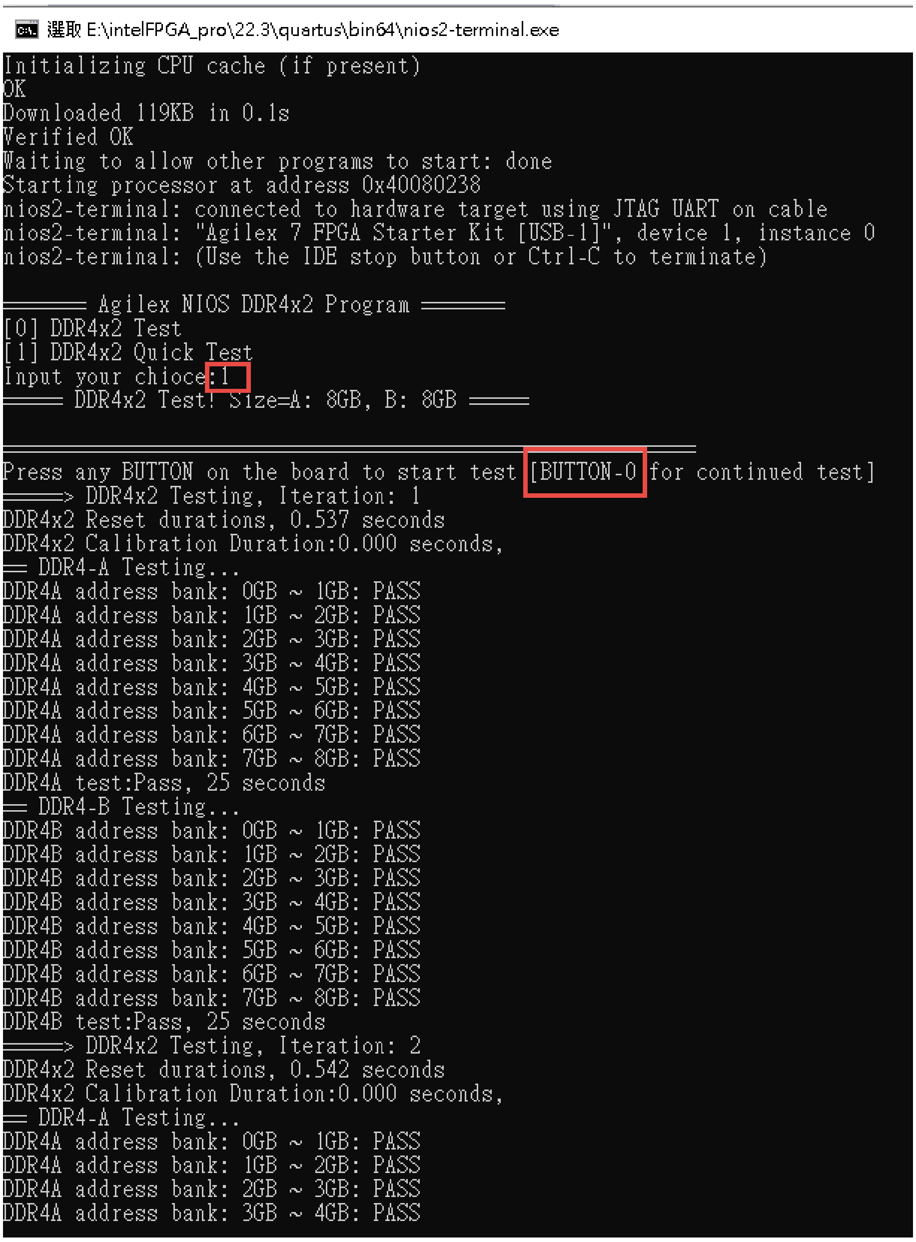


```
E:\intelFPGA_pro\22.3\quartus\bin64\nios2-terminal.exe
Downloaded 119KB in 0.1s
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x40080238
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Agilex 7 FPGA Starter Kit [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

======= Agilex NIOS DDR4x2 Program =======
[0] DDR4x2 Test
[1] DDR4x2 Quick Test
Input your chioce:0
===== DDR4x2 Test! Size=A: 8GB, B: 8GB =====

==========================================================
Press any BUTTON on the board to start test [BUTTON-0 for continued test]
=====> DDR4x2 Testing, Iteration: 1
DDR4x2 Reset durations, 0.534 seconds
DDR4x2 Calibration Duration:0.000 seconds,
== DDR4-A Testing...
DDR4 address bank: 0GB ~ 1GB:
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
DDR4 address bank: 1GB ~ 2GB:
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60% _
```

Figure 2-7 Progress option [0] DDR4x2 Test



```
選取 E:\intelFPGA_pro\22.3\quartus\bin64\nios2-terminal.exe
Initializing CPU cache (if present)
OK
Downloaded 119KB in 0.1s
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x40080238
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)

====== Agilex NIOS DDR4x2 Program ======
[0] DDR4x2 Test
[1] DDR4x2 Quick Test
Input your chioce:1
==== DDR4x2 Test! Size=A: 8GB, B: 8GB ====
===================================================
Press any BUTTON on the board to start test [BUTTON-0 for continued test]
=====> DDR4x2 Testing, Iteration: 1
DDR4x2 Reset durations, 0.537 seconds
DDR4x2 Calibration Duration:0.000 seconds,
== DDR4-A Testing...
DDR4A address bank: 0GB ~ 1GB: PASS
DDR4A address bank: 1GB ~ 2GB: PASS
DDR4A address bank: 2GB ~ 3GB: PASS
DDR4A address bank: 3GB ~ 4GB: PASS
DDR4A address bank: 4GB ~ 5GB: PASS
DDR4A address bank: 5GB ~ 6GB: PASS
DDR4A address bank: 6GB ~ 7GB: PASS
DDR4A address bank: 7GB ~ 8GB: PASS
DDR4A test:Pass, 25 seconds
== DDR4-B Testing...
DDR4B address bank: 0GB ~ 1GB: PASS
DDR4B address bank: 1GB ~ 2GB: PASS
DDR4B address bank: 2GB ~ 3GB: PASS
DDR4B address bank: 3GB ~ 4GB: PASS
DDR4B address bank: 4GB ~ 5GB: PASS
DDR4B address bank: 5GB ~ 6GB: PASS
DDR4B address bank: 6GB ~ 7GB: PASS
DDR4B address bank: 7GB ~ 8GB: PASS
DDR4B test:Pass, 25 seconds
=====> DDR4x2 Testing, Iteration: 2
DDR4x2 Reset durations, 0.542 seconds
DDR4x2 Calibration Duration:0.000 seconds,
== DDR4-A Testing...
DDR4A address bank: 0GB ~ 1GB: PASS
DDR4A address bank: 1GB ~ 2GB: PASS
DDR4A address bank: 2GB ~ 3GB: PASS
DDR4A address bank: 3GB ~ 4GB: PASS
```

Figure 2-8 Progress and Result Information for "DDR4 Quick Test"

2.4 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 A7SK board.

The A7SK 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 2-9**, 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.

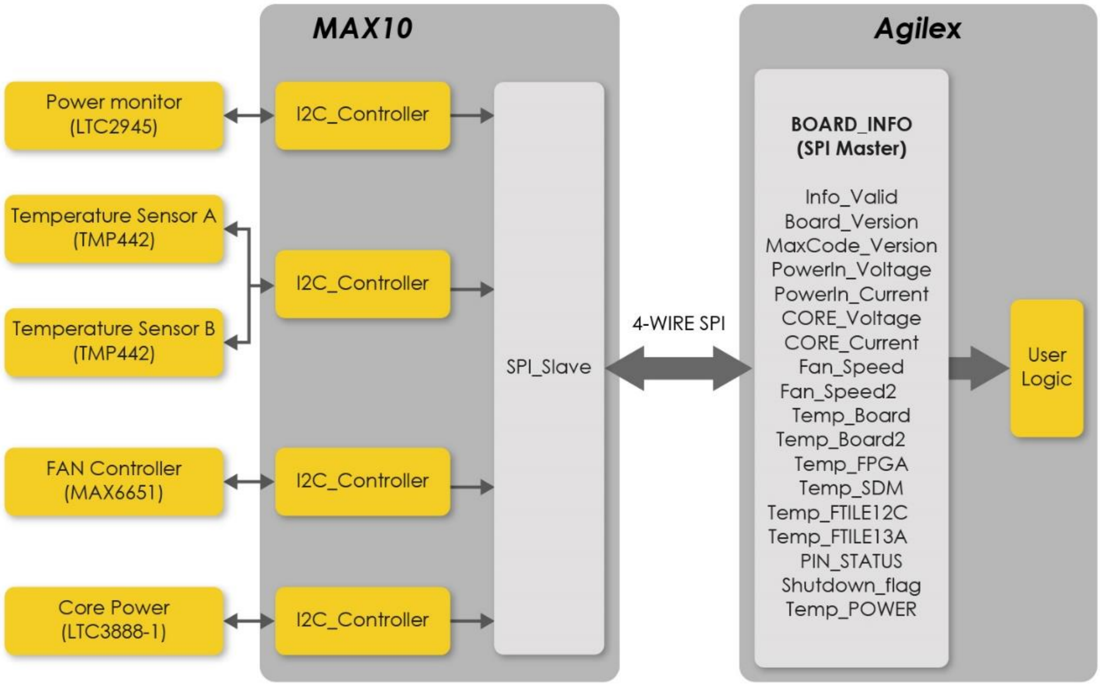


Figure 2-9 Block diagram of the fan speed control demonstration

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/spi_master/board_information_ip/BOARD_INFO.v***

Figure 2-10 shows the input and output pins of the board information IP. Detailed pin descriptions and functions can be obtained from **Table 2-3** 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 boar 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 2-11** shows the status of the IP during execution.

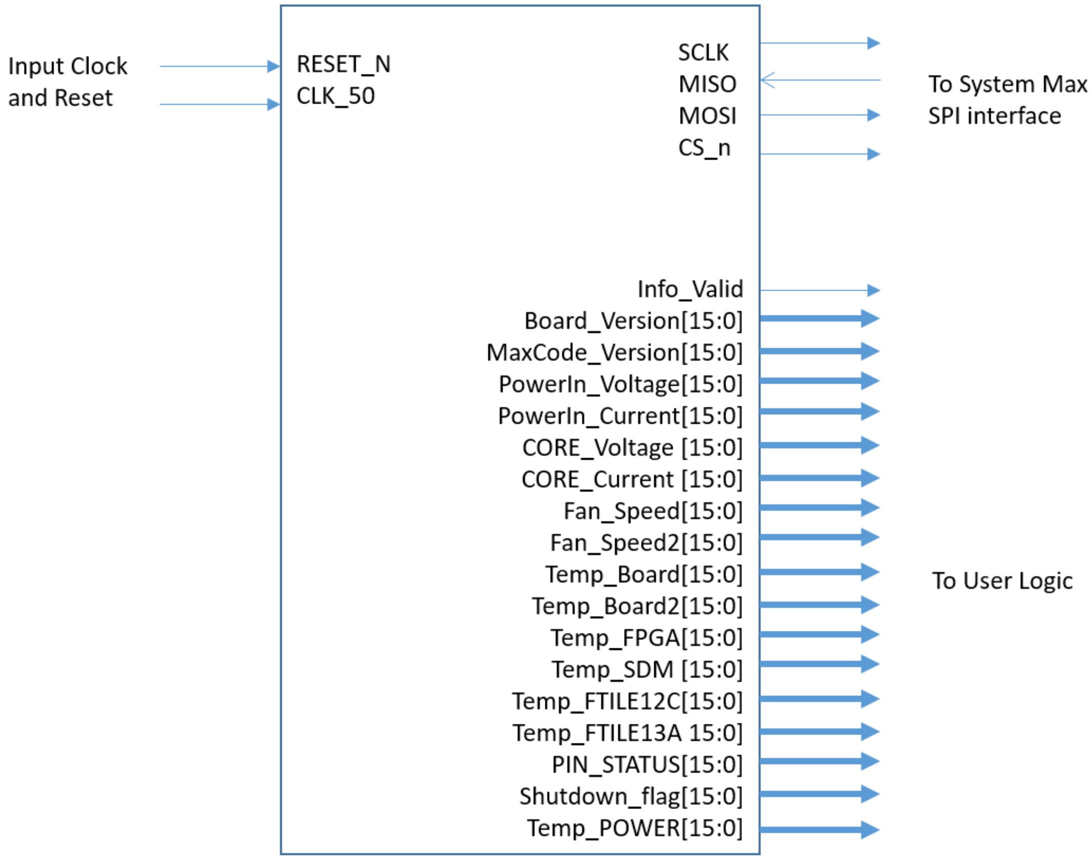


Figure 2-10 Pin out of the board information IP

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 salve. 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 A7SK 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
PowerIn_Current	Output	16	Current of the 12V power, the unit of the output value is mA. If the **PowerIn_Current**
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
Fan_Speed	Output	16	First fan speed of the board. The unit of the output value is RPM.
Fan_Speed2	Output	16	Second 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.
Temp_SDM	Output	16	SDM FPGA temperature of the development board. The unit of the output value is Celsius
Temp_FTILE12C	Output	16	Temperature of the FTILE12C transceiver in the FPGA. The unit of the output value is Celsius
Temp_FTILE13A	Output	16	Temperature of the FTILE13A transceiver in the FPGA. The unit of the output value is Celsius

Temp_POWER	Output	16	Temperature of the LTC3888. The unit of the output value is Celsius
Shutdown_flag	Output	16	BIT8~15 : Reserved to 0. BIT7: 1 ,when CORE_Current >= 100A BIT6:1 , when 12V_Power in>=160W BIT5:1 , when FPGA Temperature >= 95˚C BIT4:1 , when FTILE13A Temperature >=95˚C BIT3:1 , when Board2 Temperature >=95˚C BIT2:1 , when FTILE12C Temperature >=95˚C BIT1:1 , when SDM Temperature >=95˚C BIT0:1 , when Board1 Temperature >=95˚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. BIT 2:**FPGA_CONF_DONE**,FPGA Configure success, this bit is 1. bit1: Reserved to 1. bit0: Reserved to 1.

Table 2-3 Board information IP input and output ports

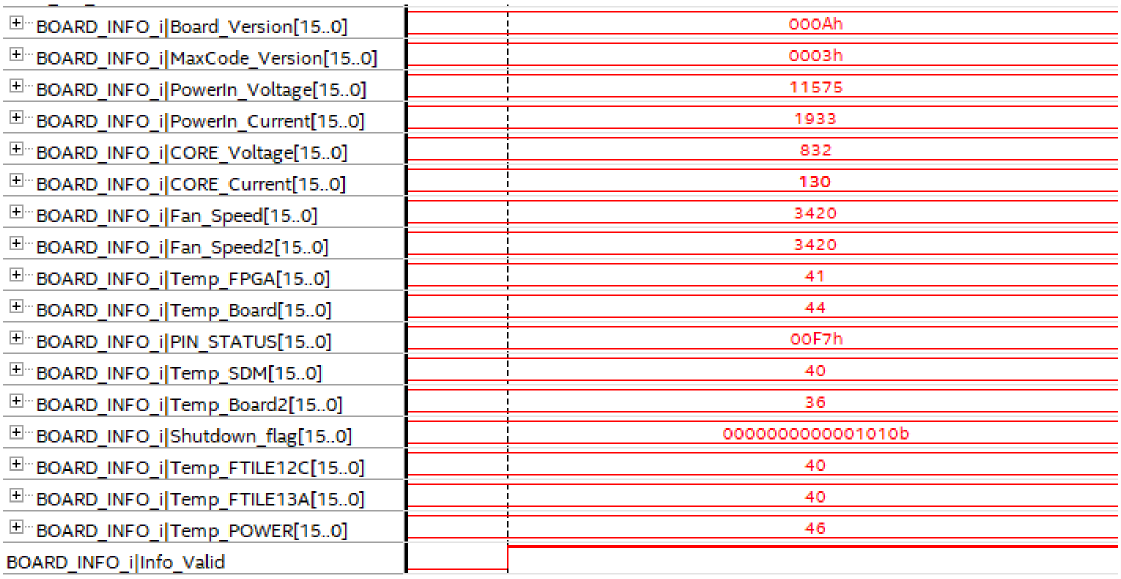


Figure 2-11 Waveform of the board status output

2.5 100G Ethernet Example

This 100G Ethernet example is generated according to the documents F-Tile Ethernet Intel FPGA Hard IP Design Example User Guide. The F-Tile Ethernet Hard 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 2-12** shows the block diagram of this demonstration.

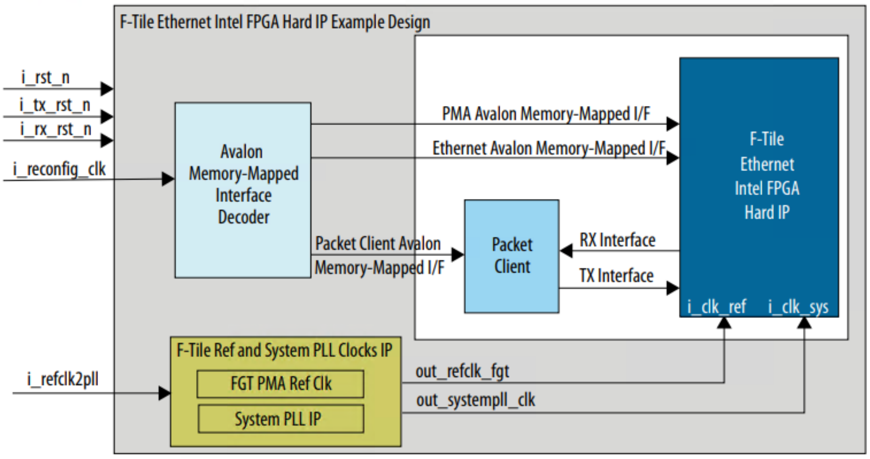


Figure 2-12 Block diagram of 100GbE demo

■ Project Information

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

Item	Description
Project Location	Ethernet_100G
Quartus Project	Ethernet_100G\hardware_test_design
FPGA Bit Stream	Ethernet_100G\ hardware_test_design\output_files
Test Scrip File	Ethernet_100G\hardware_test_design\hwtest\main_100G.tcl
Quartus Version	Quartus Prime 23.2 Pro Edition

Figure 2-13 shows the IP setup for the demonstration. **100GE-4** Ethernet mode and **IEEE 802.3 RS(528,514)(CL 91)** are selected.

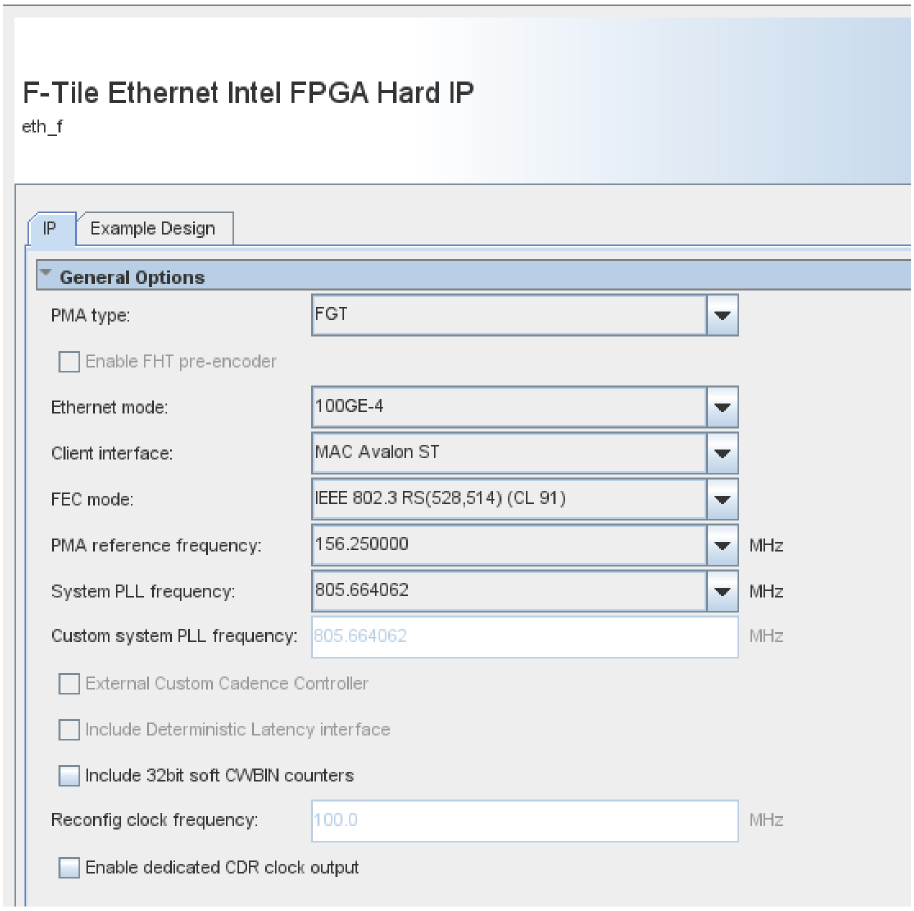


Figure 2-13 100G Setup for Ethernet IP

■ Demonstration Setup

Here is the procedure to setup the demonstration. A QSFP28 loopback fixtures are required for external loopback. If you don't have the loopback fixture, please use **run_test** instead of **run_test_without_loopback** in the following demonstration procedure. The **run_test** enables transceiver serial loopback for internal loopback.

1. Insert a QSFP28 loopback fixture into the QSFP28 port on the Agilex FPGA board, as shown in **Figure 2-14**.
2. Connect the host PC to the FPGA board using a micro-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 program the "**eth_f_hw.sof**" to the FPGA.

4. Open eth_f_hw 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 2-15**.

   %cd hwtest

   %source main_100G.tcl

   %run_test_without_loopback

6. The loopback test report will be displayed in the Tcl Console, as shown in **Figure 2-16**.

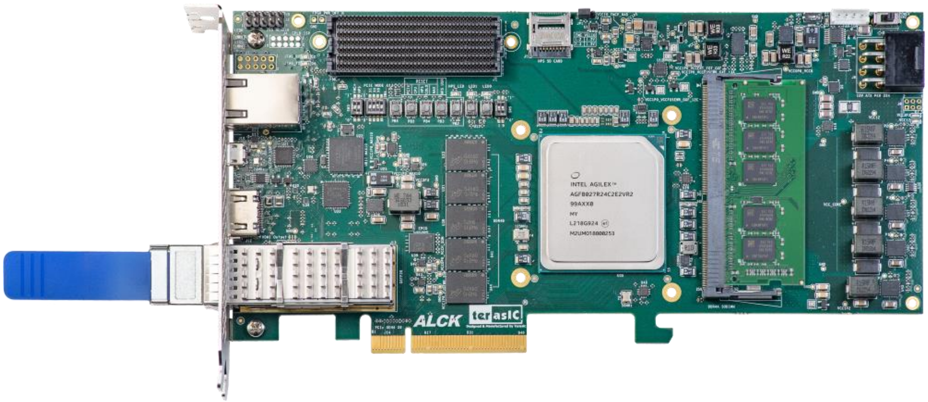


Figure 2-14 Setup QSFP28 loopback fixture

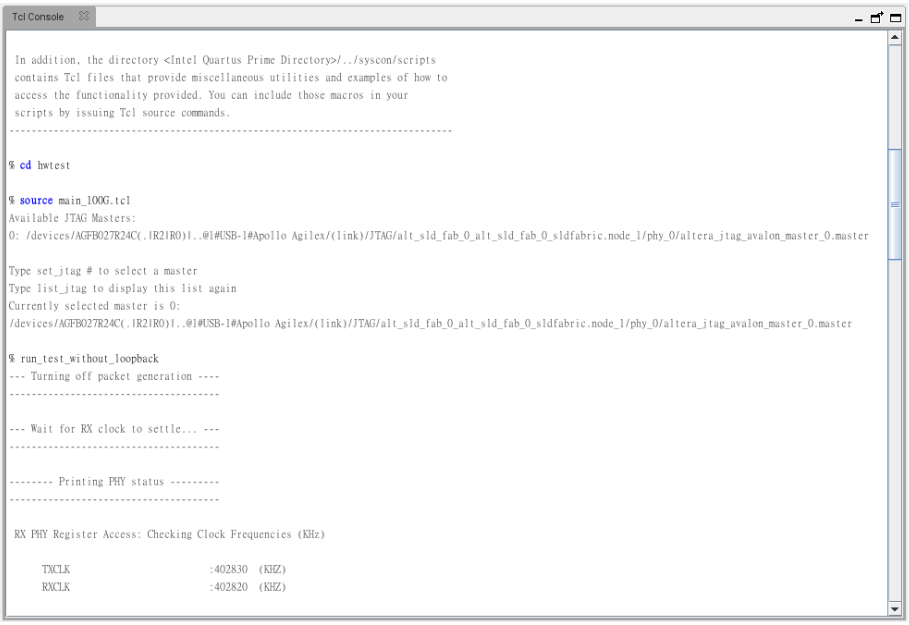


Figure 2-15 Launch the System Console for Ethernet 100G Demo

```
% cd hwtest

% source main_100G.tcl
Available JTAG Masters:
0: /devices/AGFB027R24C(.|B|R2|R0)|..@1#USB-1#Agilex 7 FPGA Starter
Kit/(link)/JTAG/alt_sld_fab_0_alt_sld_fab_0_sldfabric.node_1/phy_0/altera_jtag_avalon_master_0.master

Type set_jtag # to select a master
Type list_jtag to display this list again
Currently selected master is 0:
/devices/AGFB027R24C(.|B|R2|R0)|..@1#USB-1#Agilex 7 FPGA Starter
Kit/(link)/JTAG/alt_sld_fab_0_alt_sld_fab_0_sldfabric.node_1/phy_0/altera_jtag_avalon_master_0.master

% run_test_without_loopback
--- Turning off packet generation ----
--------------------------------------

--- Wait for RX clock to settle... ---
--------------------------------------

-------- Printing PHY status ---------
--------------------------------------

 RX PHY Register Access: Checking Clock Frequencies (KHz)

 Note: For i_reconfig_clk other than 100MHz, Clock Frequency readout below must be scaled for display
      TXCLK                         :402820  (KHZ)
      RXCLK                         :402840  (KHZ)


 TX PLL Lock Status           0x0000000f

 Rx Frequency Lock Status     0x0000000f

 RX PCS Ready                 0x1

 TX Lanes Stable              0x1

 Deskewed Status              0x1

 Link Fault Status            0x00000000

 Rx Frame Error               0x00000000

 Rx AM LOCK Condition         0x1

---- Clearing MAC stats counters -----
--------------------------------------

--- Initialize PKT ROM Read address for IP_INST[0]----
--------------------------------------
```

Figure 2-16 Ethernet 100G loopback test report for RX

```
==================================================================
                 STATISTICS FOR BASE 0x3000 (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                    : 0
65 - 127 Byte Frames              : 16
128 - 255 Byte Frames             : 0
256 - 511 Byte Frames             : 0
512 - 1023 Byte Frames            : 0
1024 - 1518 Byte Frames           : 0
1519 - MAX Byte Frames            : 0
> MAX Byte Frames                 : 0
Rx Frame Starts                   : 16
Multicast data  OK  Frame         : 16
Broadcast data OK   Frame         : 0
Unicast data OK   Frames          : 0
Multicast Control Frames          : 0
Broadcast Control Frames          : 0
Unicast Control Frames            : 0
Pause Control Frames              : 0
Data and padding octets           : 864
Frame octets                      : 1152
```

Figure 2-17 Ethernet 100G loopback test report for TX

```
==========================================================
                 STATISTICS FOR BASE 0x3000 (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                    : 0
65 - 127 Byte Frames              : 16
128 - 255 Byte Frames             : 0
256 - 511 Byte Frames             : 0
512 - 1023 Byte Frames            : 0
1024 - 1518 Byte Frames           : 0
1519 - MAX Byte Frames            : 0
> MAX Byte Frames                 : 0
Tx Frame Starts                   : 16
Multicast data  OK  Frame         : 16
Broadcast data OK   Frame         : 0
Unicast data OK   Frames          : 0
Multicast Control Frames          : 0
Broadcast Control Frames          : 0
Unicast Control Frames            : 0
Pause Control Frames              : 0
Data and padding octets           : 864
Frame octets                      : 1152
--------------------------------------

run_test:pass
--------------------------------------

---------------- Done ---------------
```

Figure 2-18 Ethernet 100G loopback test summary

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 3-19 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.

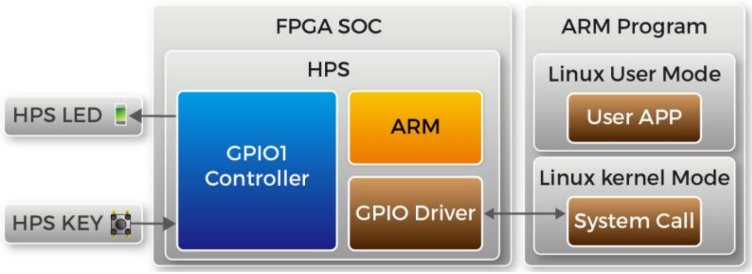


Figure 3-19 Function block diagram of HPS LED/KEY demonstration

■ 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 3-20 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 3-20 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 3-21**.
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: 0
key: 1
key: 1
^C
terasic@localhost:~$
```

Figure 3-21 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 3-22**. 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.

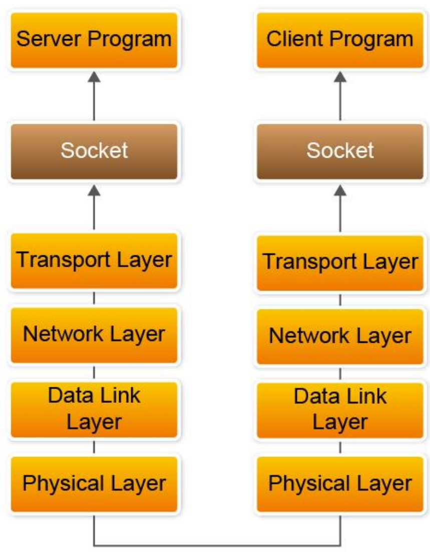


Figure 3-22 Communicate on a network via a socket

■ 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 3-23** shows the communication diagram between the client and server.

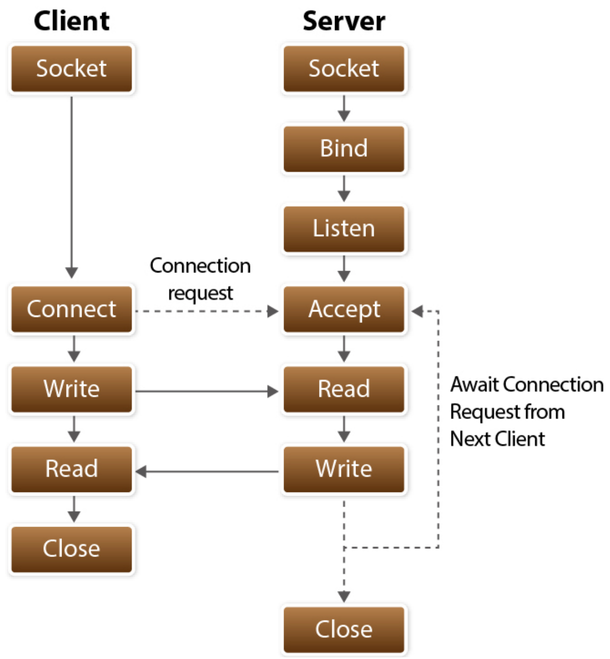


Figure 3-23 Client and Server communication

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

API header file – socket.h.

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 3-24** 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 3-24 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 3-25** 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 IP4v 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.s_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 3-25 Socket Client Code

■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-26**. 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.

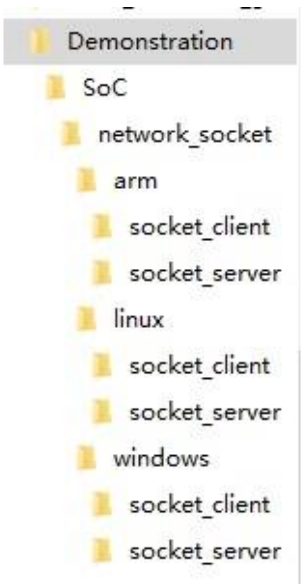


Figure 3-26 Source Code Folder Tree

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 to 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 3-27**. The port number can be any value between 2000 and 63500.

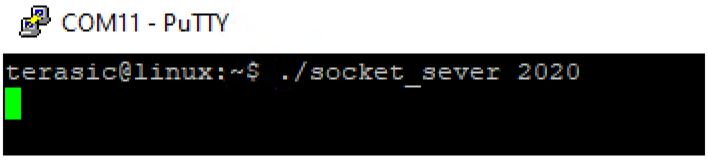


Figure 3-27 Start Socket Server

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 3-28**.

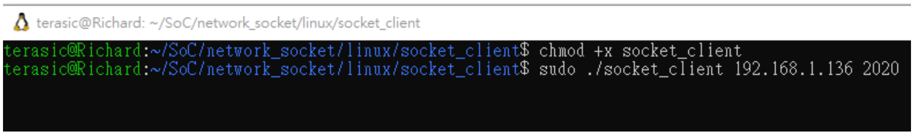


Figure 3-28 Start Client Program

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 3-29**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-30**.

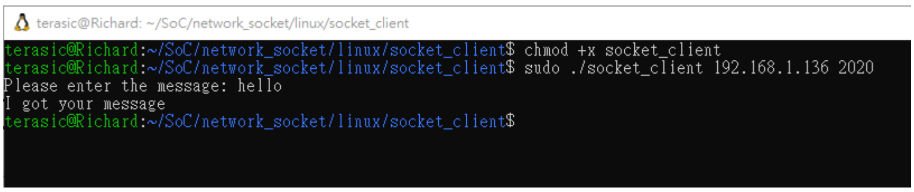


Figure 3-29 Send Message in Client Program

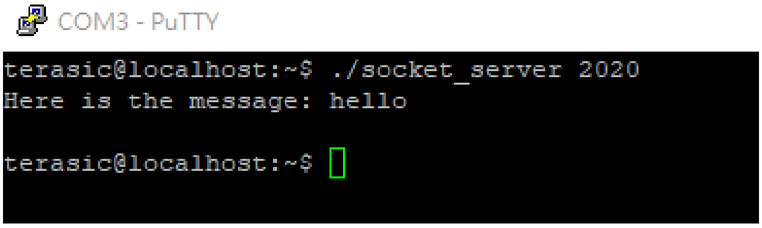


Figure 3-30 Server dumps received message

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 3-31.**

```
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 3-31 Build C/C++ Project

Chapter 4

PCI Express Reference Design for Windows

PCI Express is commonly used in consumer, server, and industrial applications, to link motherboard-mounted peripherals. From this demonstration, it will show how the PC Windows and FPGA communicate with each other through the PCI Express interface. Multi Channel DMA Intel® FPGA IP for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel document ug20297-683821-750934.

4.1 PCI Express System Infrastructure

Figure 4-1 shows the infrastructure of the PCI Express System in this demonstration. It consists of two primary components: FPGA System and PC System. The FPGA System is developed based on Multi Channel DMA Intel® FPGA IP for PCI Express. The application software on the PC side is developed by Terasic based on Intel's PCIe kernel mode driver.

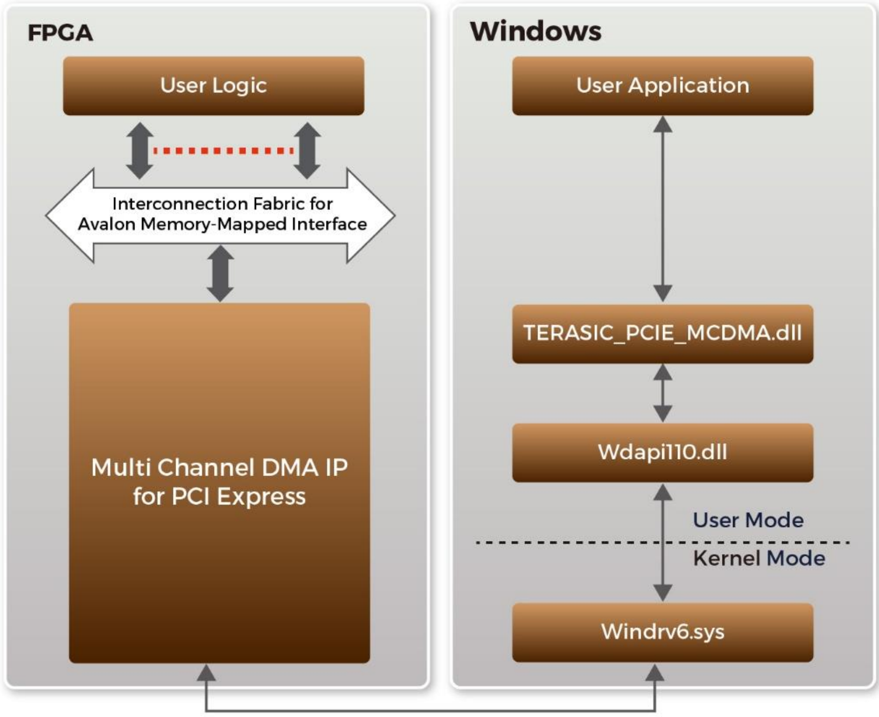


Figure 4-1 Infrastructure of PCI Express System

4.2 PC PCI Express Software SDK

The FPGA System CD contains a PC Windows based SDK to allow users to develop their 64-bit software application on 64-bits Windows 10. The SDK is located in the "CDROM\Demonstrations\PCIe_SW_KIT\Windows" folder which includes:

- PCI Express Driver
- PCI Express Library
- PCI Express Examples

The kernel mode driver assumes the PCIe vendor ID (VID) is 0x1172 and the device ID (DID) is 0x09C4. If different VID and DID are used in the design, users need to modify the PCIe vendor ID (VID) and device ID (DID) in the driver INF file accordingly.

The PCI Express Library is implemented as a single DLL named TERASIC_PCIE_MCDMA.dll. This file is a 64-bit DLL. When the DLL is exported to the software API, users can easily communicate with the FPGA. The library provides the following functions:

- Basic data read and write
- Data read and write by DMA

For high performance data transmission, MCDMA is required as the read and write operations, which are specified under the hardware design on the FPGA.

4.3 PCI Express Software Stack

Figure 4-2 shows the software stack for the PCI Express application software on 64-bit Windows. The PCIe library module TERASIC_PCIE_MCDMA.dll provides DMA and direct I/O access allowing user application program to communicate with FPGA. Users can develop their applications based on this DLL. The altera_pcie_win_driver.sys kernel driver is provided by Altera.

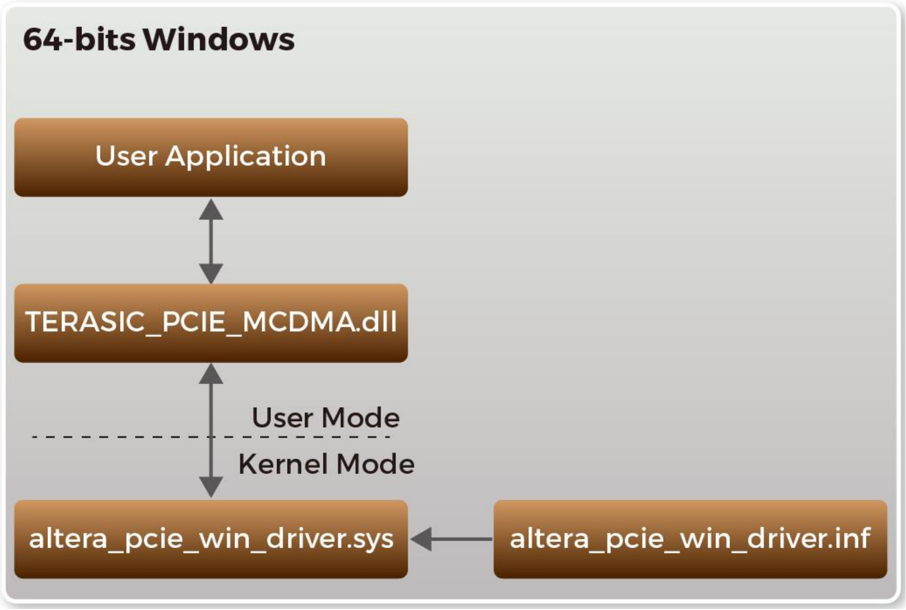


Figure 4-2 PCI Express Software Stack

■ Install PCI Express Driver on Windows

The PCIe driver is locate in the folder:

"CDROM\Demonstrations\PCIe_SW_KIT\Windows\PCIe_Driver"

The folder includes the following four files:

- Altera_pcie_win_driver.cat
- Altera_pcie_win_driver.inf
- Altera_pcie_win_driver.sys
- WdfCoinstaller01011.dll

To install the PCI Express driver, please execute the steps below:

1. Install the board on the PCIe slot of the host PC.
2. Make sure the Intel Programmer and USB-Blaster II driver are installed.
3. Because the windows 10 enforces driver signatures by default and the OpenCL drivers for our development kits are not "signed" for Windows 10. So, for windows10, please refer to the link to disable driver signature enforcement and reboot system.
4. Execute test.bat in "CDROM\Demonstrations\PCIe_Fundamental\demo_batch" to configure the FPGA.
5. Restart windows operation system.
6. Click the Control Panel menu from Windows Start menu. Click the Hardware and Sound item before clicking the Device Manager to launch the Device Manager dialog. There will be a PCI Device item in the dialog, as shown in **Figure 4-3**. Move the mouse cursor to the PCI Device item and right click it to select the Updated Driver Software... items.

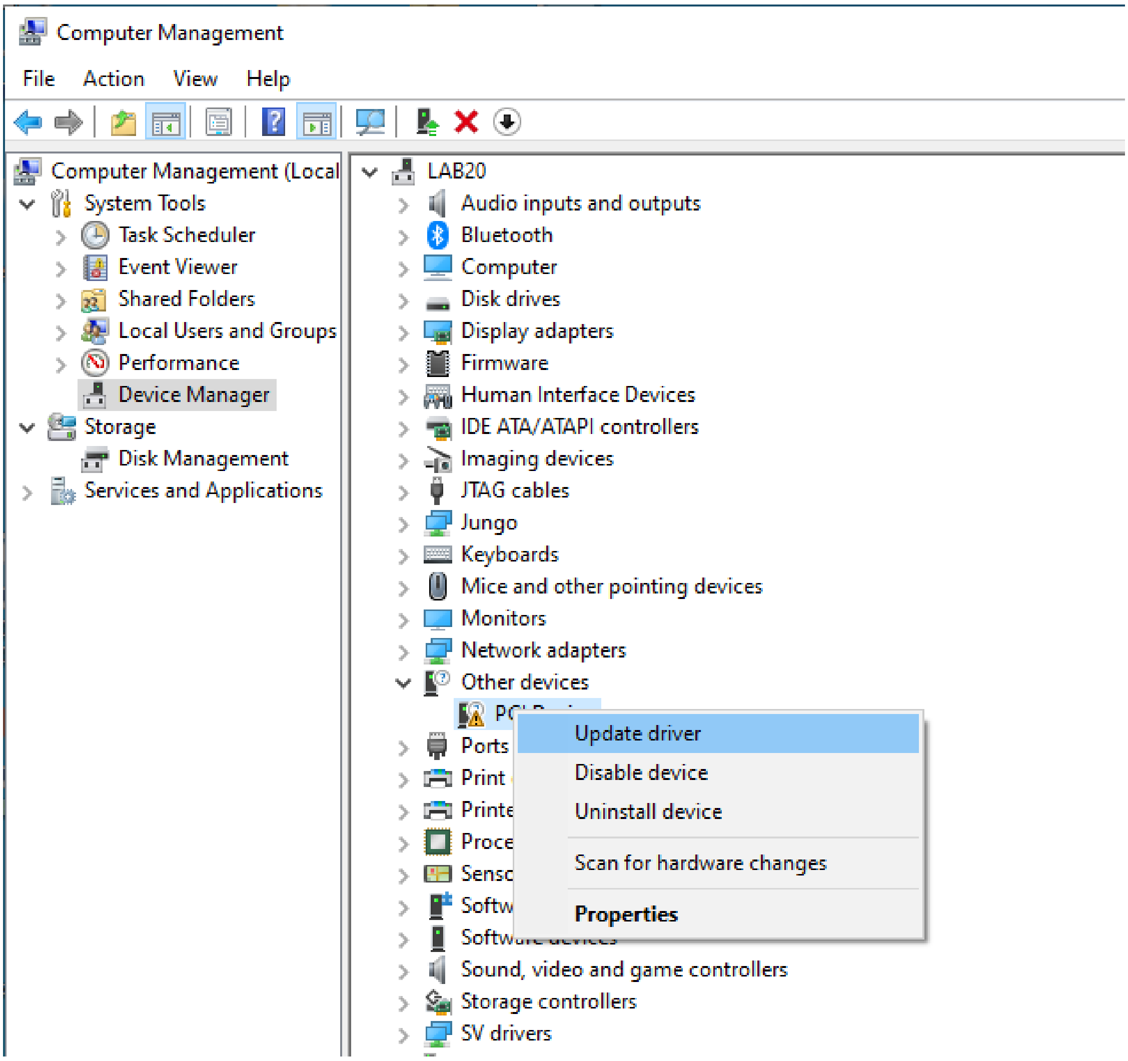


Figure 4-3 Screenshot of launching Update Driver Software… dialog

7. In the **How do you want to search for the driver software** dialog, click **Browse my computer for driver software** item, as shown in **Figure 4-4**

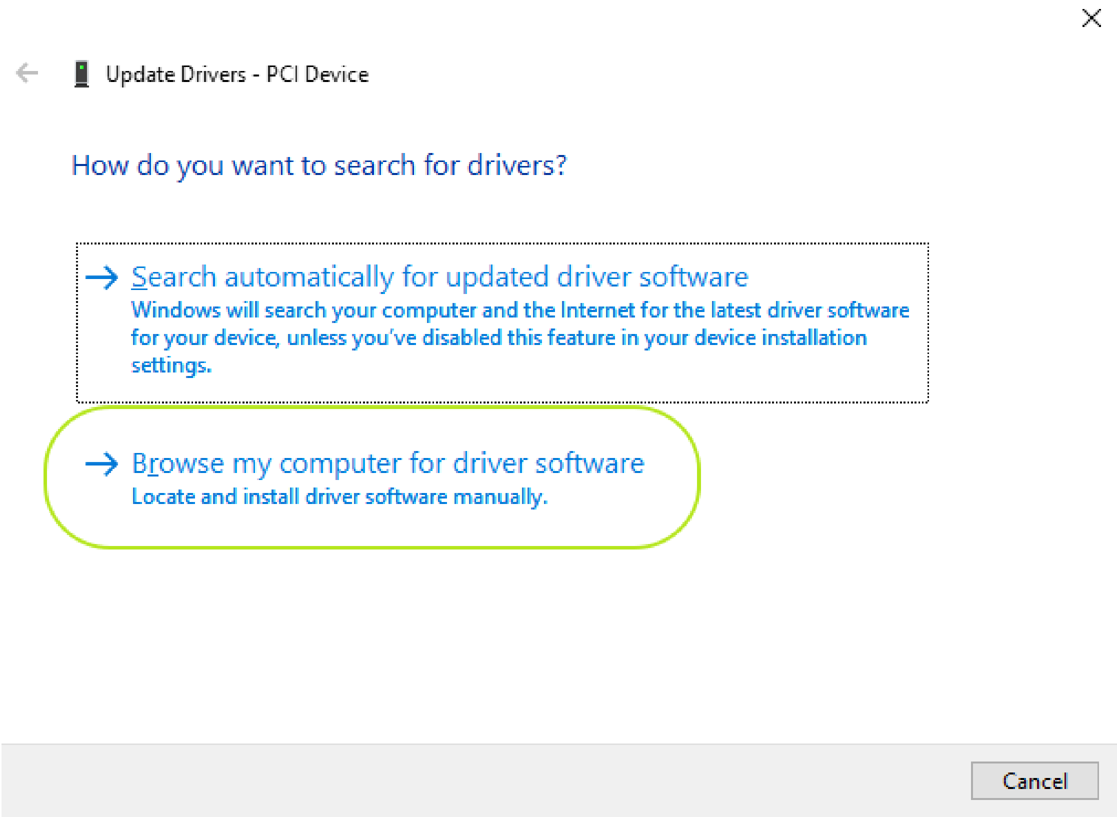


Figure 4-4 Dialog of Browse my computer for the driver software

8. In the **Browse for driver software on your computer** dialog, click the **Browse** button to specify the folder where altera_pcie_din_driver.inf is located, as shown in **Figure 4-5**. Click the **Next** button.

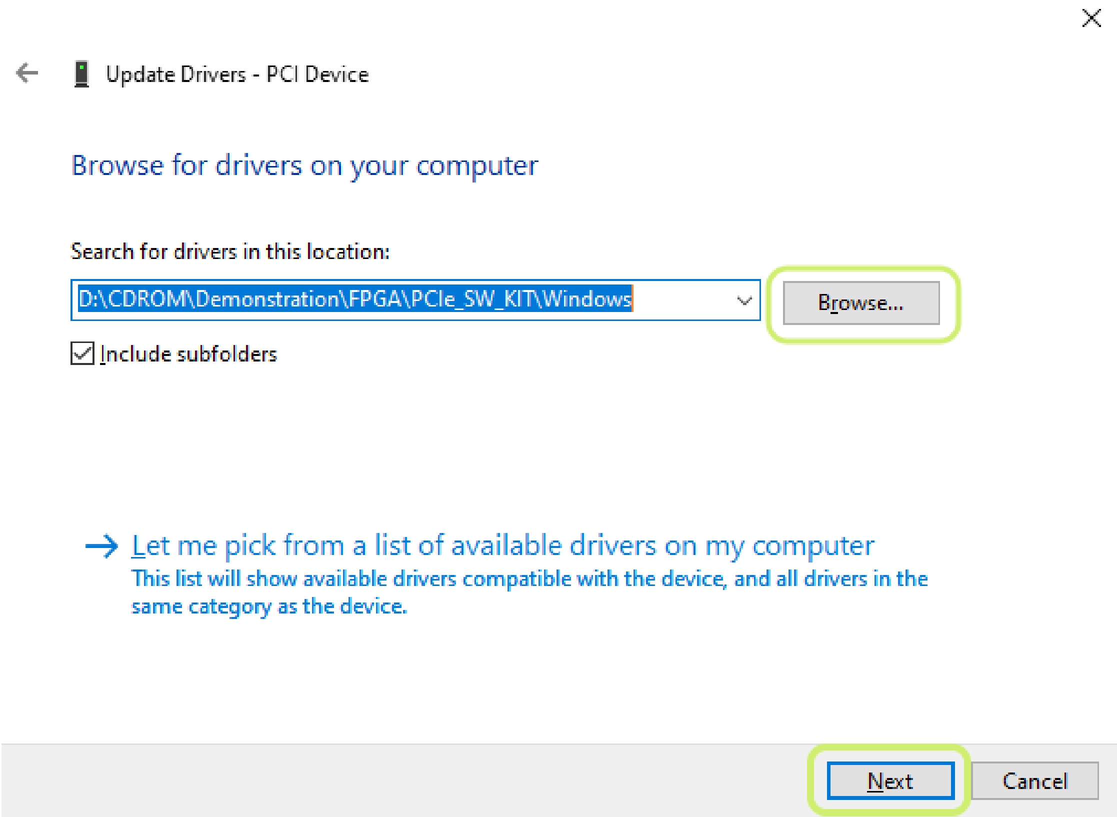


Figure 4-5 Browse for the driver software on your computer

9. When the **Windows Security** dialog appears, as shown **Figure 4-6**, click the **Install** button.

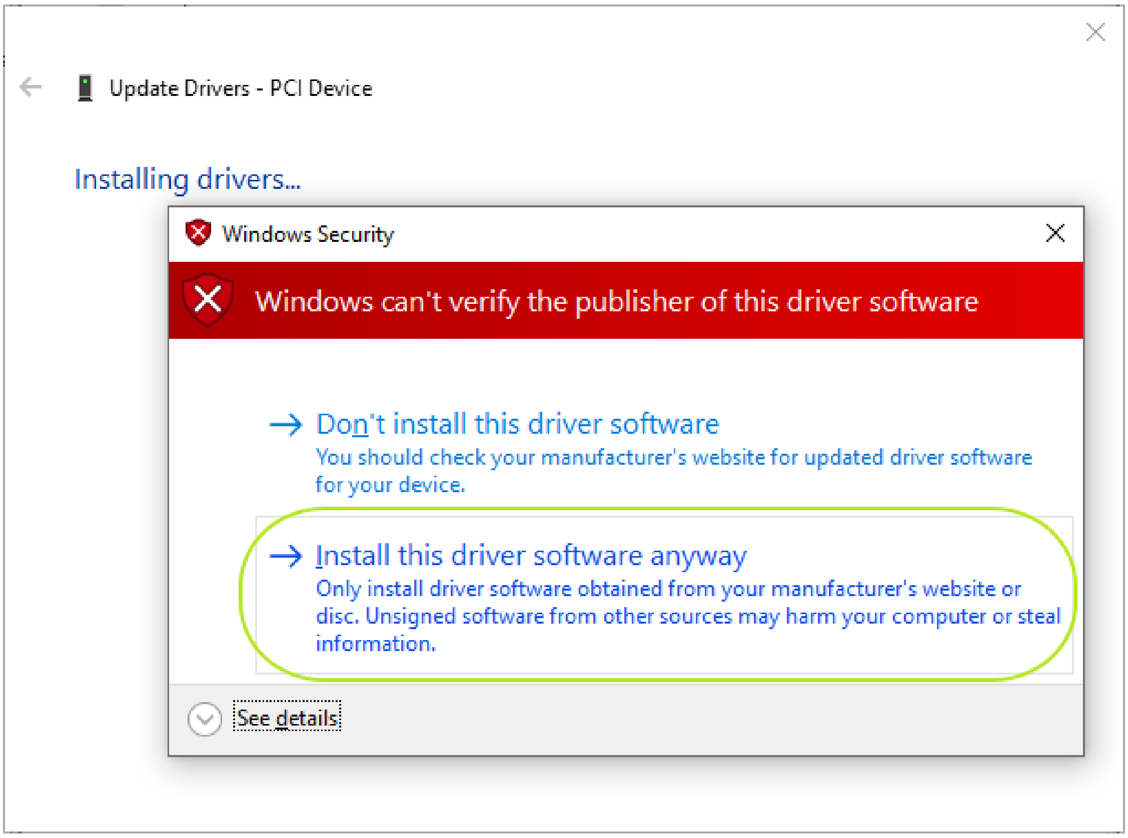


Figure 4-6 Click Install in the dialog of Windows Security

10. When the driver is installed successfully, the successfully dialog will appear, as shown in **Figure 4-7**. Click the **Close** button.

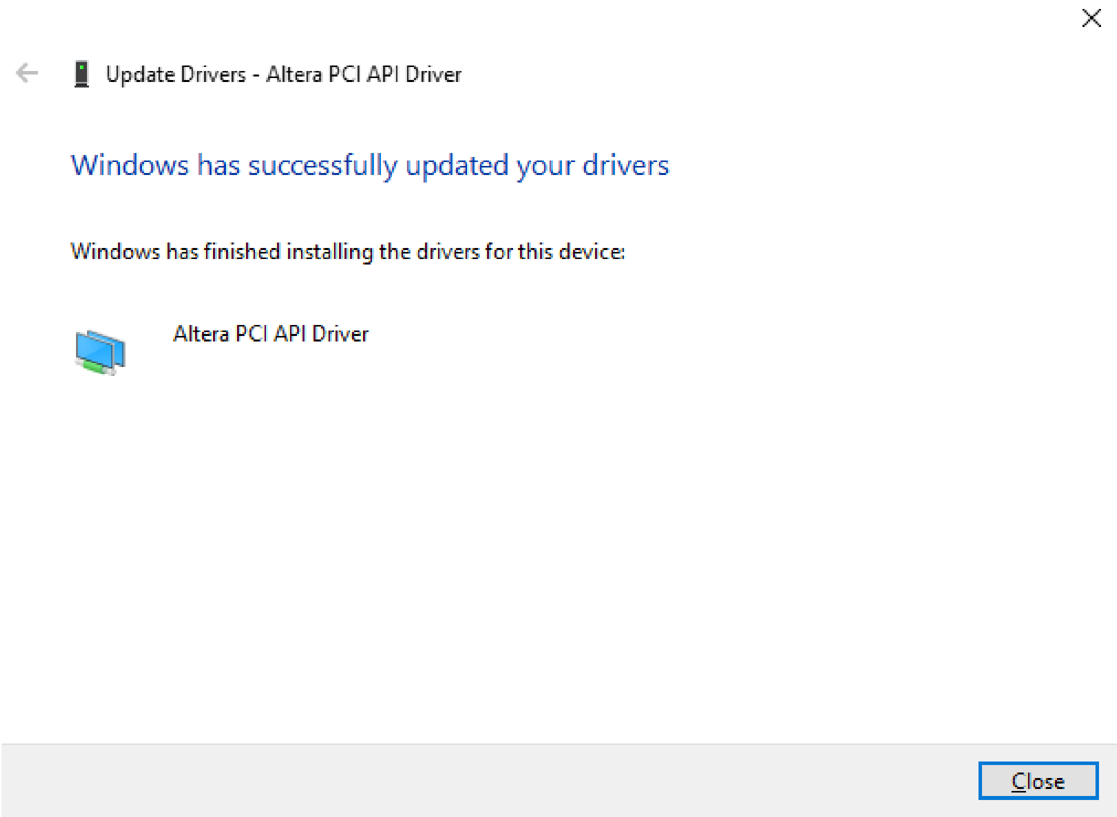


Figure 4-7 Click Close when the installation of the Altera PCI API Driver is complete

11. Once the driver is successfully installed, users can see the **Altera PCI API Driver** under the device manager window, as shown in **Figure 4-8**.

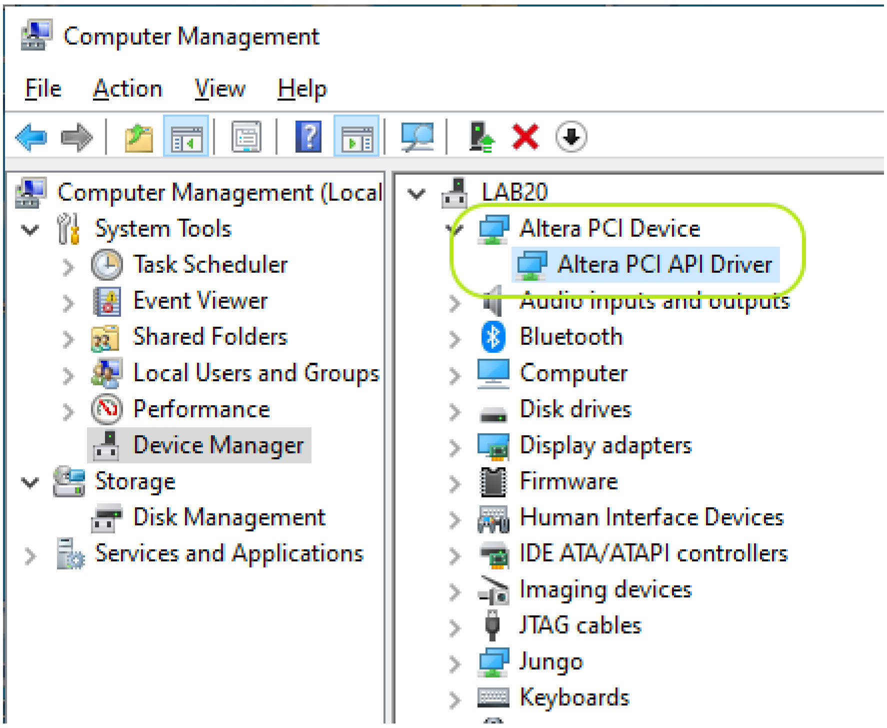


Figure 4-8 Altera PCI API Driver in Device Manager

■ Create a Software Application

All the files needed to create a PCIe software application are located in the directory CDROM\demonstration\PCIe_SW_KIT\Windows\PCIe_Library. It includes the following files:

- TERASIC_PCIE_MCDMA.h
- TERASIC_PCIE_MCDMA.dll (64-bit DLL)

Below lists the procedures to use the SDK files in users' C/C++ project:

1. Create a 64-bit C/C++ project.
2. Include TERASIC_PCIE_MCDMA.h in the C/C++ project.
3. Copy TERASIC_PCIE_MCDMA.dll to the folder where the project.exe is located.
4. Dynamically load TERASIC_PCIE_MCDMA.dll in C/C++ program. To load the DLL, please refer to the PCIe fundamental example below.
5. Call the SDK API to implement the desired application.

Users can easily communicate with the FPGA through the PCIe bus through the TERASIC_PCIE_MCDMA.dll API. The details of API are described below:

4.4 PCI Express Library API

Below shows the exported API in the TERASIC_PCIE_MCDMA.dll. The API prototype is defined in the TERASIC_PCIE_MCDMA.h.

Note: the Linux library terasic_pcie_mcdma.so also use the same API and header file.

■ PCIE_Open

Function: Open a specified PCIe card with vendor ID, device ID, and matched card index.
Prototype: PCIE_HANDLE PCIE_Open(uint8_t wVendorID, uint8_t wDeviceID, uint8_t wCardIndex);
Parameters: wVendorID: Specify the desired vendor ID. A zero value means to ignore the vendor ID. wDeviceID: Specify the desired device ID. A zero value means to ignore the device ID. wCardIndex: Specify the matched card index, a zero based index, based on the matched vendor ID and device ID.
Return Value: Return a handle to presents specified PCIe card. A positive value is return if the PCIe card is opened successfully. A value zero means failed to connect the target PCIe card. This handle value is used as a parameter for other functions, e.g. PCIE_Read32. Users need to call PCIE_Close to release handle once the handle is no longer used.

■ PCIE_Close

Function: Close a handle associated to the PCIe card.
Prototype: void PCIE_Close(

PCIE_HANDLE hPCIE);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function.
Return Value: None.

■ PCIE_Read32

Function: Read a 32-bit data from the FPGA board.
Prototype: bool PCIE_Read32(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, uint32_t *pdwData);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function. PcieBar: Specify the target BAR. PcieAddress: Specify the target address in FPGA. pdwData: A buffer to retrieve the 32-bit data.
Return Value: Return **true** if read data is successful; otherwise **false** is returned.

■ PCIE_Write32

Function: Write a 32-bit data to the FPGA Board.
Prototype: bool PCIE_Write32(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar,

PCIE_ADDRESS PcieAddress,
uint32_t dwData);

Parameters:

hPCIE:

A PCIe handle return by PCIE_Open function.

PcieBar:

Specify the target BAR.

PcieAddress:

Specify the target address in FPGA.

dwData:

Specify a 32-bit data which will be written to FPGA board.

Return Value:

Return **true** if write data is successful; otherwise **false** is returned.

■ PCIE_Read8

Function:

Read an 8-bit data from the FPGA board.

Prototype:

```
bool PCIE_Read8(
    PCIE_HANDLE hPCIE,
    PCIE_BAR PcieBar,
    PCIE_ADDRESS PcieAddress,
    uint8_t *pByte);
```

Parameters:

hPCIE:

A PCIe handle return by PCIE_Open function.

PcieBar:

Specify the target BAR.

PcieAddress:

Specify the target address in FPGA.

pByte:

A buffer to retrieve the 8-bit data.

Return Value:

Return **true** if read data is successful; otherwise **false** is returned.

■ PCIE_Write8

Function: Write an 8-bit data to the FPGA Board.
Prototype: bool PCIE_Write8(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, uint8_t Byte);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function. PcieBar: Specify the target BAR. PcieAddress: Specify the target address in FPGA. Byte: Specify an 8-bit data which will be written to FPGA board.
Return Value: Return **true** if write data is successful; otherwise **false** is returned.

■ PCIE_DmaRead

Function: Read data from the memory-mapped memory of FPGA board in DMA.
Prototype: bool PCIE_DmaRead(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pBuffer, uint64_t dwBufSize64);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function. LocalAddress:

Specify the target memory-mapped address in FPGA. pBuffer: A pointer to a memory buffer to retrieved the data from FPGA. The size of buffer should be equal or larger the dwBufSize. dwBufSize64: Specify the byte number of data retrieved from FPGA.
Return Value: Return **true** if read data is successful; otherwise **false** is returned.

■ PCIE_DmaWrite

Function: Write data to the memory-mapped memory of FPGA board in DMA.
Prototype: bool PCIE_DmaWrite(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, uint64_t dwDataSize64);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function. LocalAddress: Specify the target memory mapped address in FPGA. pData: A pointer to a memory buffer to store the data which will be written to FPGA. dwDataSize64: Specify the byte number of data which will be written to FPGA.
Return Value: Return **true** if write data is successful; otherwise **false** is returned.

■ PCIE_ConfigRead32

Function: Read PCIe Configuration Table. Read a 32-bit data by given a byte offset.
Prototype:

bool PCIE_ConfigRead32 (PCIE_HANDLE hPCIE, uint32_t Offset, uint32_t *pdwData);
Parameters: hPCIE: A PCIe handle return by PCIE_Open function. Offset: Specify the target byte of offset in PCIe configuration table. pdwData: A 4-bytes buffer to retrieve the 32-bit data.
Return Value: Return **true** if read data is successful; otherwise **false** is returned.

4.5 PCIe Reference Design - Fundamental

The application reference design shows how to implement fundamental control and data transfer in DMA. In the design, basic I/O is used to control the BUTTON and LED on the FPGA board. High-speed data transfer is performed by the DMA.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_Fundamental\demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_Fundamental.sof
- Download Batch file: test.bat
- Windows Application Software folder: windows_app, includes
  - PCIE_FUNDAMENTAL.exe
  - TERASIC_PCIE_MCDMA.dll

■ Demonstration Setup

1. Install the FPGA board on your PC as shown in **Figure 4-9**.

Figure 4-9 FPGA board installation on PC

2. Configure FPGA with PCIe_Fundamental.sof by executing the test.bat.
3. Install the PCIe driver if necessary. The driver is located in the folder:
   CDROM\Demonstration\PCIe_SW_KIT\Windows\PCIe_Driver.
4. Restart Windows
5. Make sure that Windows has detected the FPGA Board by checking the Windows Device Manager as shown in **Figure 4-10**.

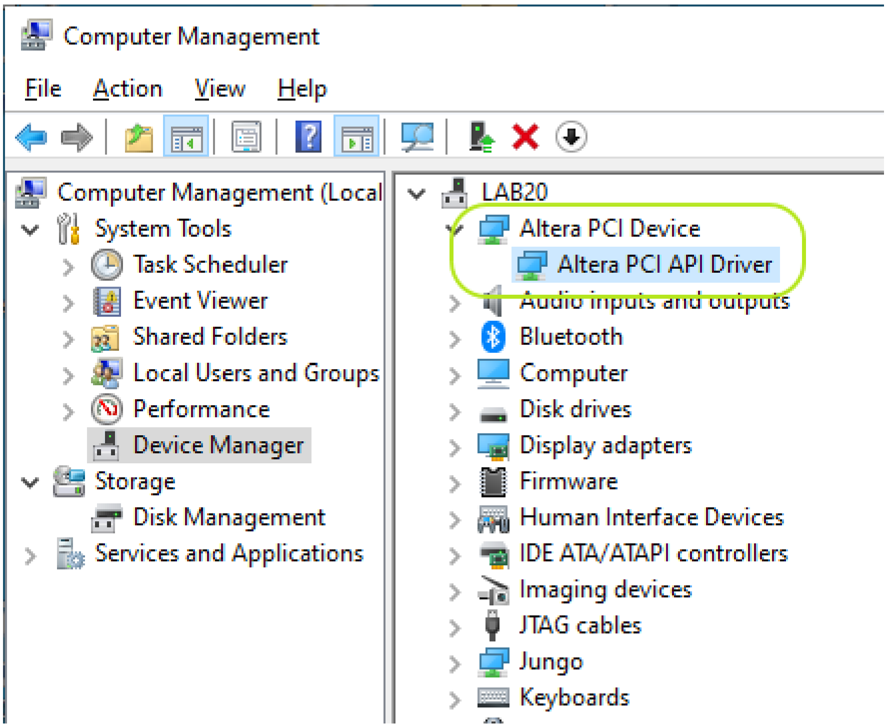


Figure 4-10 Screenshot for PCIe Driver

6. Go to windows_app folder, execute PCIE_FUNDMENTAL.exe. A menu will appear as shown in **Figure 4-11**.

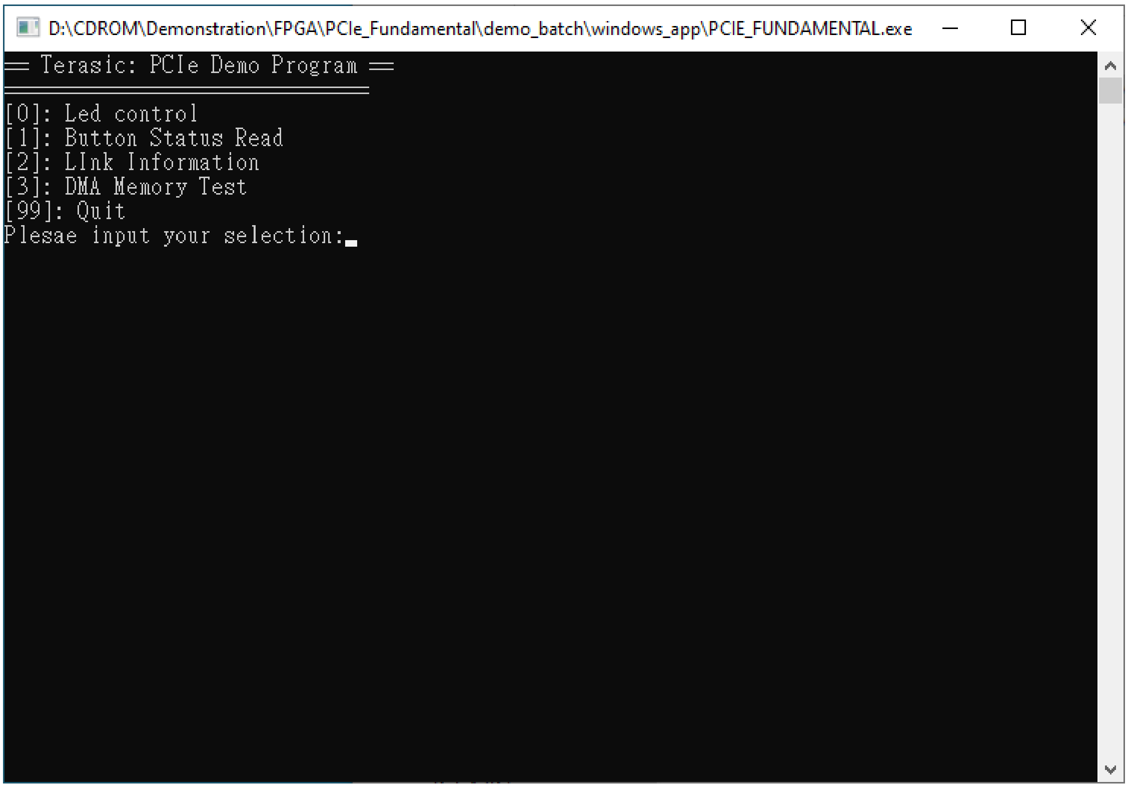


Figure 4-11 Screenshot of Program Menu

7. Type 0 followed by a ENTER key to select Led Control item, then input 3 (hex 0x03) will make all LEDs on as shown in **Figure 4-12**. If input 0 (hex 0x00), all LEDs will be turned off.

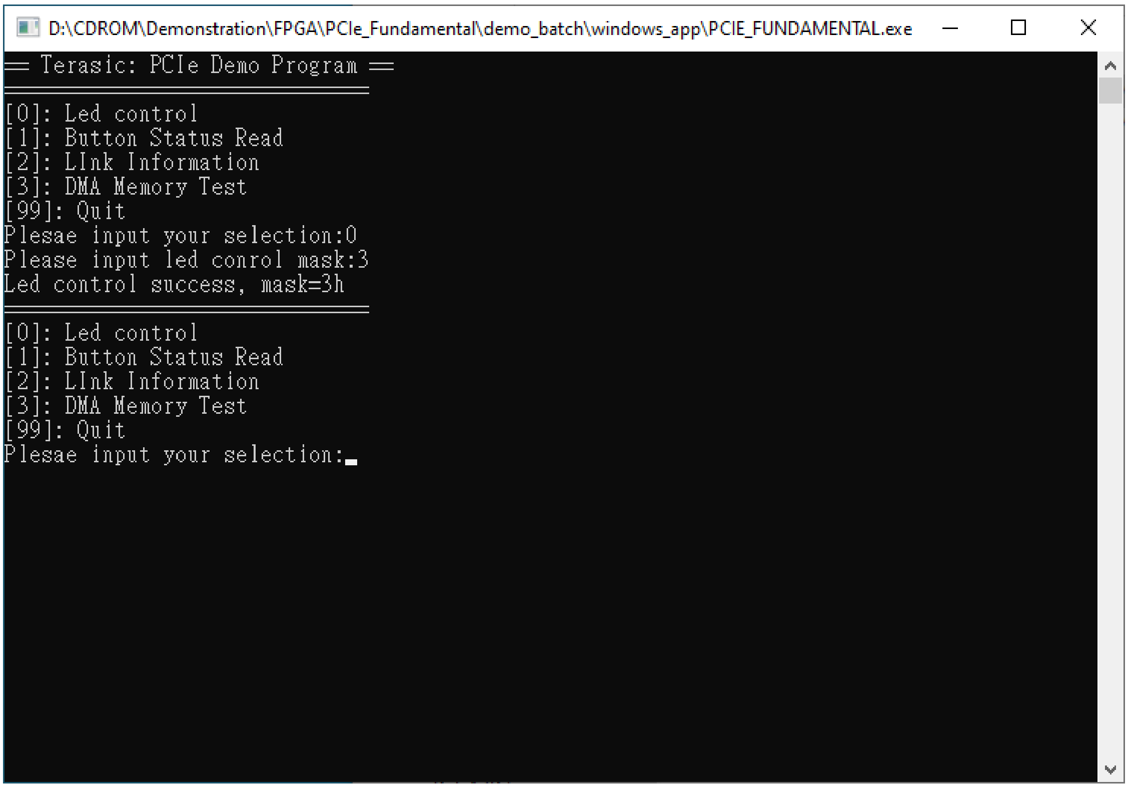


Figure 4-12 Screenshot of LED Control

8. Type 1 followed by an ENTER key to select Button Status Read item. The button status will be reported as shown in **Figure 4-13**.

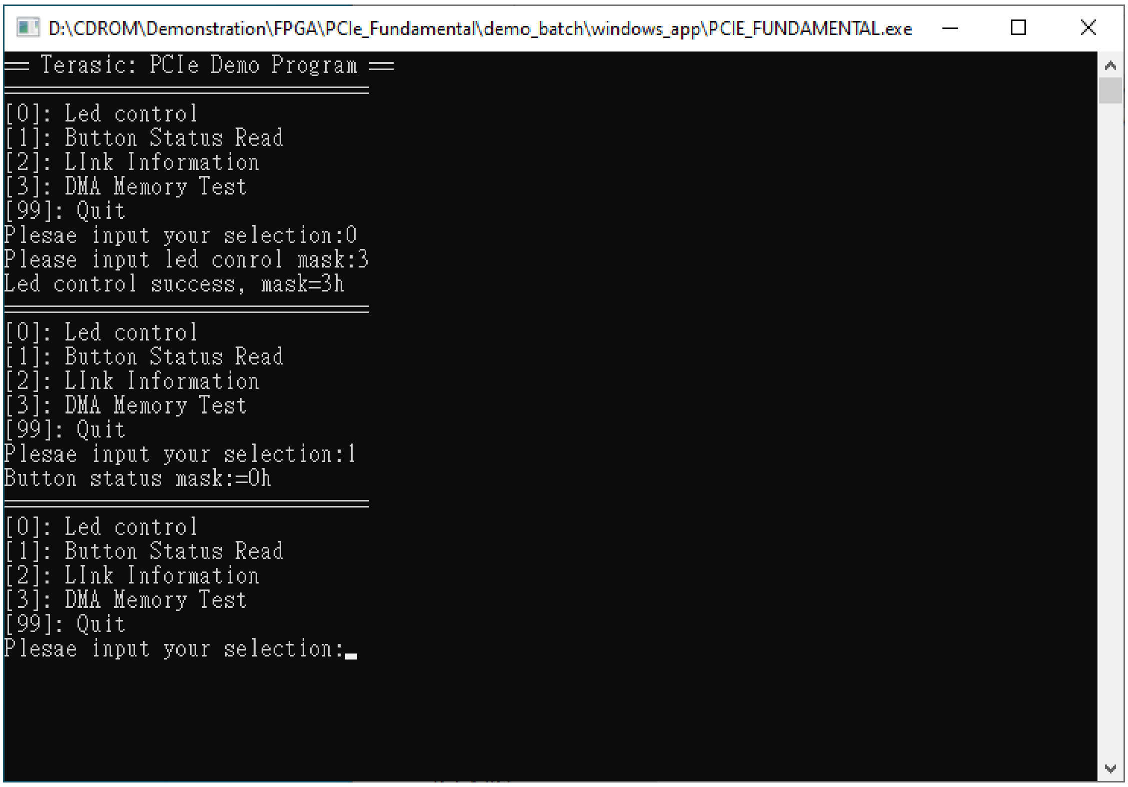


Figure 4-13 Screenshot of Button Status Report

9. Type 3 followed by an ENTER key to select the DMA Testing item. The DMA test result will be reported as shown in **Figure 4-14**.

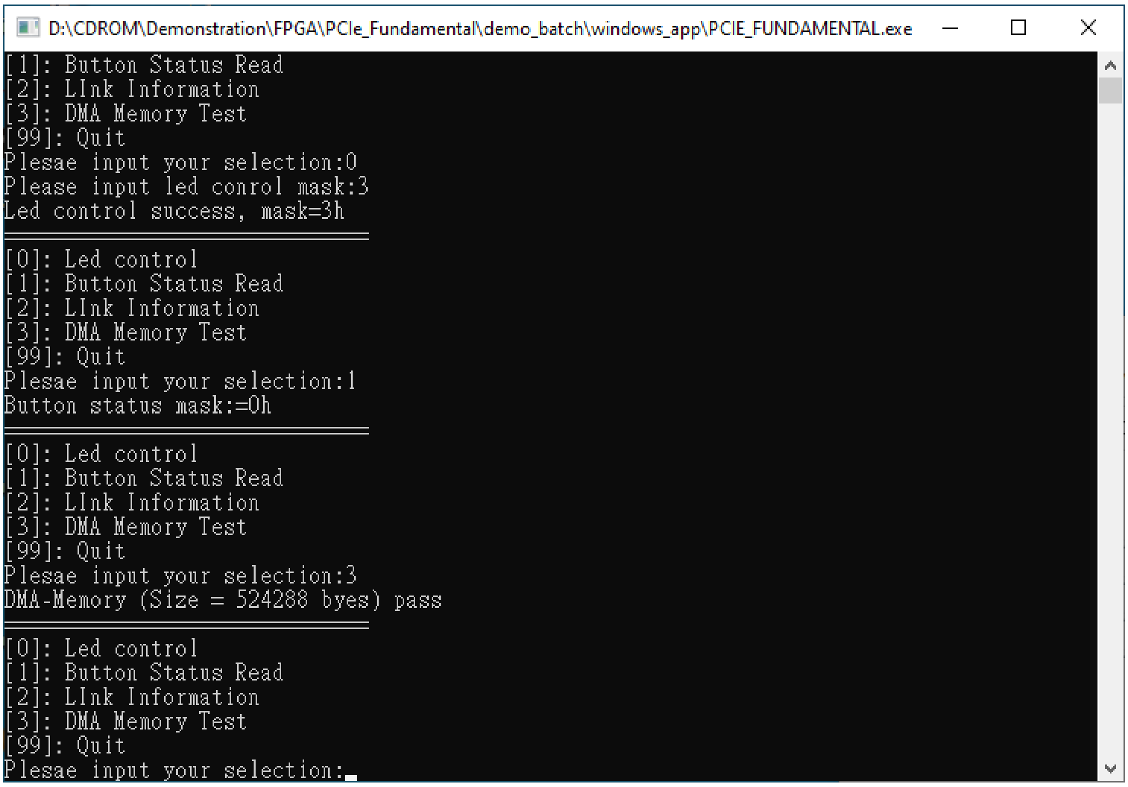


Figure 4-14 Screenshot of DMA Memory Test Result

10. Type 99 followed by an ENTER key to exit this test program

■ Development Tools

- Quartus Prime 23.2 Pro Edition
- Visual C++ 2019

■ Demonstration Source Code Location

- Quartus Project: Demonstrations\PCIe_Fundamental
- C++ Project: Demonstrations\PCIe_SW_KIT\Windows\PCIE_FUNDAMENTAL

■ FPGA Application Design

Figure 4-15 shows the system block diagram in the FPGA system. In the **Platform Designer** (formerly Qsys), the PIO controller is used to control the LED and monitor the Button Status, and the On-Chip memory is used for performing DMA testing. The PIO controllers and the On-Chip memory are connected to the PCI Express Hard IP controller through the Memory-Mapped Interface.

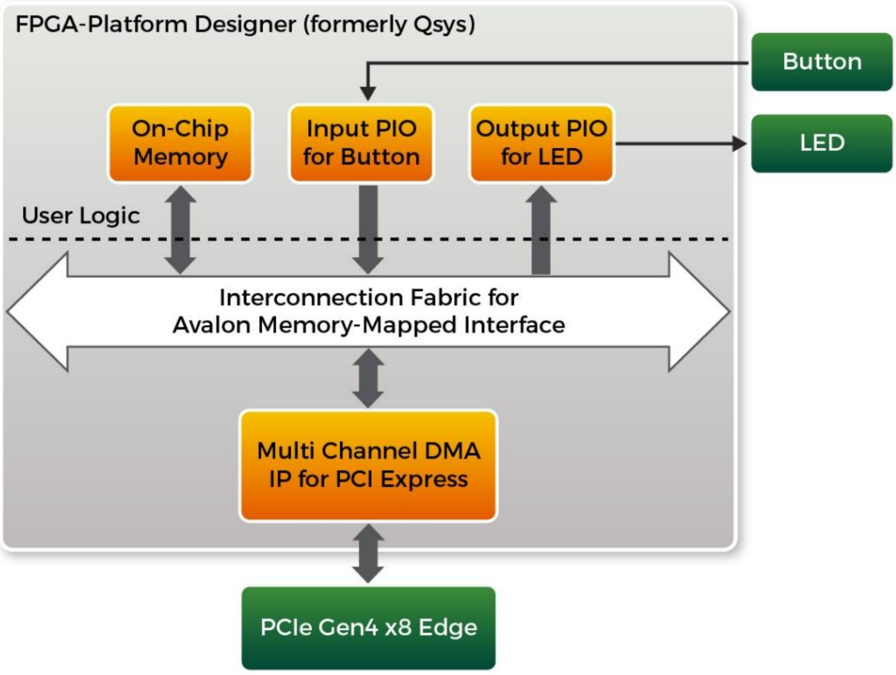


Figure 4-15 Hardware block diagram of the PCIe reference design

■ Windows Based Application Software Design

The application software project is built by Visual C++ 2019. The project includes the following major files:

Name	Description
PCIE_FUNDAMENTAL.cpp	Main program
PCIE.c	Implement dynamically load for TERASIC_PCIE_MCDMA.dll
PCIE.h	
TERASIC_PCIE_MCDMA.h	SDK library file, defines constant and data structure

The main program PCIE_FUNDAMENTAL.cpp includes the header file "PCIE.h" and defines the controller address according to the FPGA design.

```
#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x800000
#define DEMO_PCIE_IO_BUTTON_ADDR    0x800040
#define DEMO_PCIE_MEM_ADDR          0x100000

#define MEM_SIZE                    (512*1024) //512KB
```

The base address of BUTTON and LED controllers are 0x800040 and 0x800000 based on the PCIE_BAR4, respectively. The on-chip memory base address is 0x100000 relative to the DMA controller.

Before accessing the FPGA through PCI Express, the application first calls the PCIE_Load to dynamically load the TERASIC_PCIE_MCDMA.dll. Then, it calls PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in the PCIE_Open are defined in TERASIC_PCIE_MCDMA.h. If developers change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value defined in TERASIC_PCIE_MCDMA.h. If the return value of PCIE_Open is zero, it means the driver cannot be accessed successfully. In this case, please make sure:

- The FPGA is configured with the associated bit-stream file and the host is rebooted.
- The PCI express driver is loaded successfully.

The LED control is implemented by calling **PCIE_Write32** API, as shown below:

```
bPass = PCIE_Write32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_LED_ADDR, (uint32_t) Mask);
```

The button status query is implemented by calling the **PCIE_Read32** API, as shown below:

```
bPass = PCIE_Read32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_BUTTON_ADDR, &Status);
```

The memory-mapped memory read and write test is implemented by **PCIE_DmaWrite** and **PCIE_DmaRead** API, as shown below:

```
bPass = PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
bPass = PCIE_DmaRead(hPCIe, LocalAddr, pRead, nTestSize);
```

4.6 PCIe Reference Design - DDR4

The application reference design shows how to add the DDR4 Memory Controllers for the DDR4-A SODIMM and DDR4-B Component into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 8GB data DMA for both SODIMM. Also, this demo shows how to call "PCIE_ConfigRead32" API to check PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_DDR4\demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_DDR4.sof
- Download Batch file: test.bat
- Windows Application Software folder: windows_app, includes
  - ✧ PCIE_DDR4.exe
  - ✧ TERASIC_PCIE_MCDMA.dll

■ Demonstration Setup

1. Install one pieces of DDR4 8GB SODIMM on the FPGA board.
2. Install the FPGA board on your PC.
3. Configure the FPGA with the PCIe_DDR4.sof by executing the test.bat.
4. Install the PCIe driver if necessary.
5. Restart Windows
6. Make sure that Windows has detected the FPGA Board by checking the Windows Control panel.
7. Go to windows_app folder, execute PCIE_DDR4.exe. A menu will appear as shown in **Figure 4-16**.

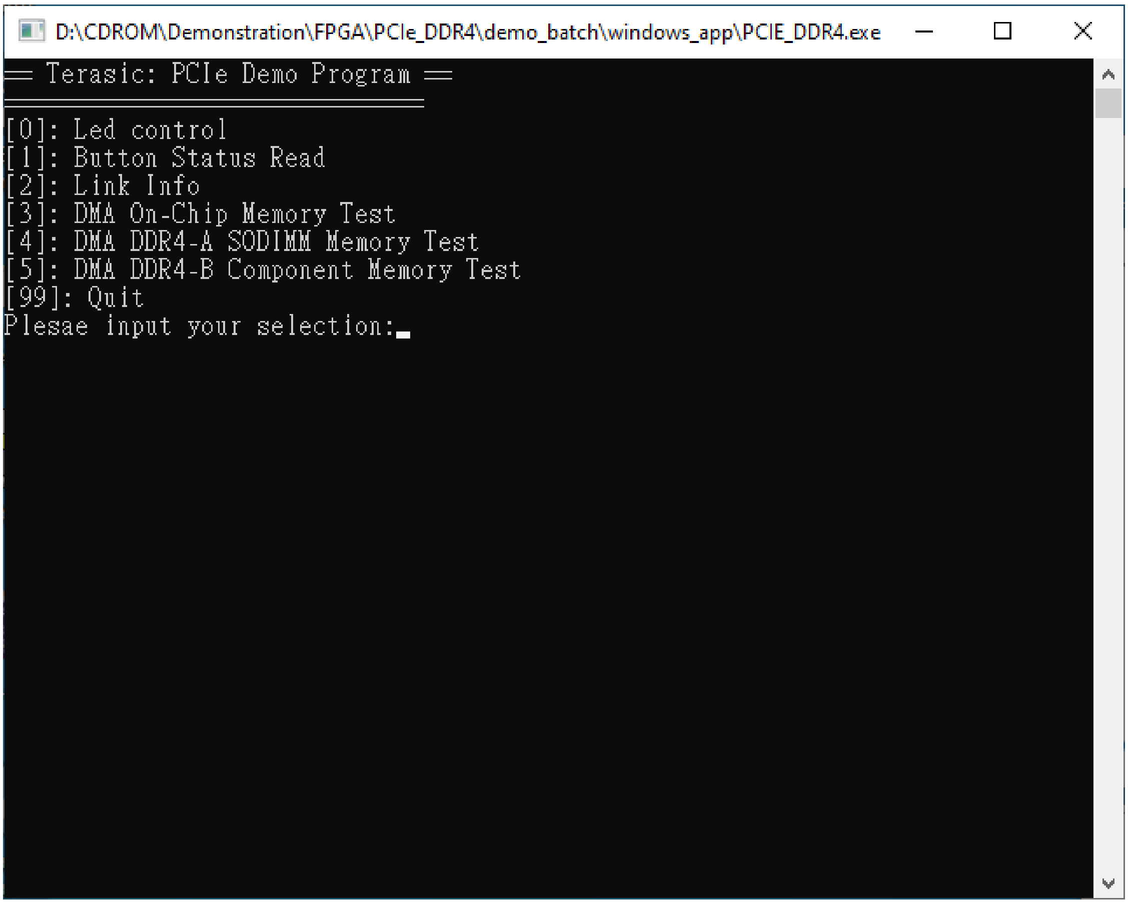


Figure 4-16 Screenshot of Program Menu

8. Type 2 followed by the ENTER key to select the Link Info item. The PCIe link information will be shown as in **Figure 4-17**. Gen4 link speed and x8 link width are expected.

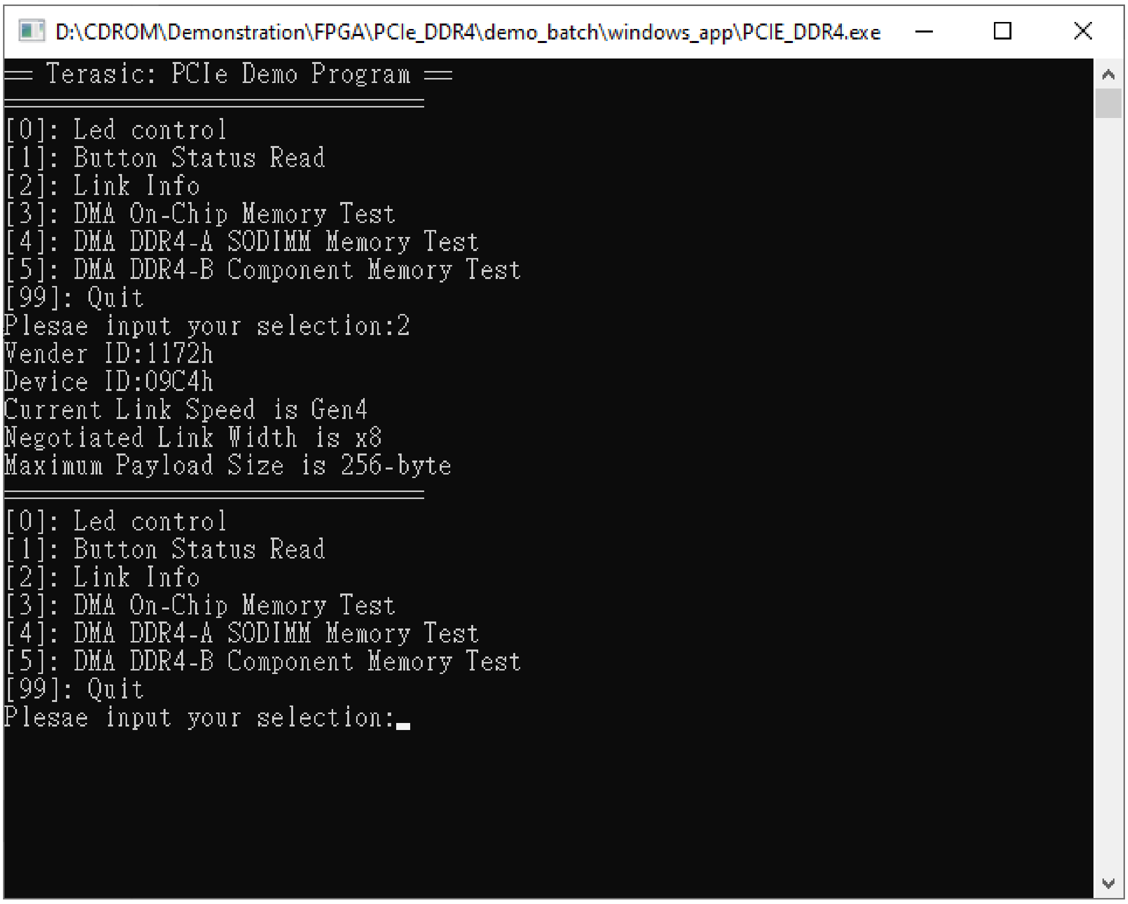


Figure 4-17 Screenshot of Link Info

9. Type 3 followed by the ENTER key to select DMA On-Chip Memory Test item. The DMA write and read test result will be reported as shown in **Figure 4-18**.

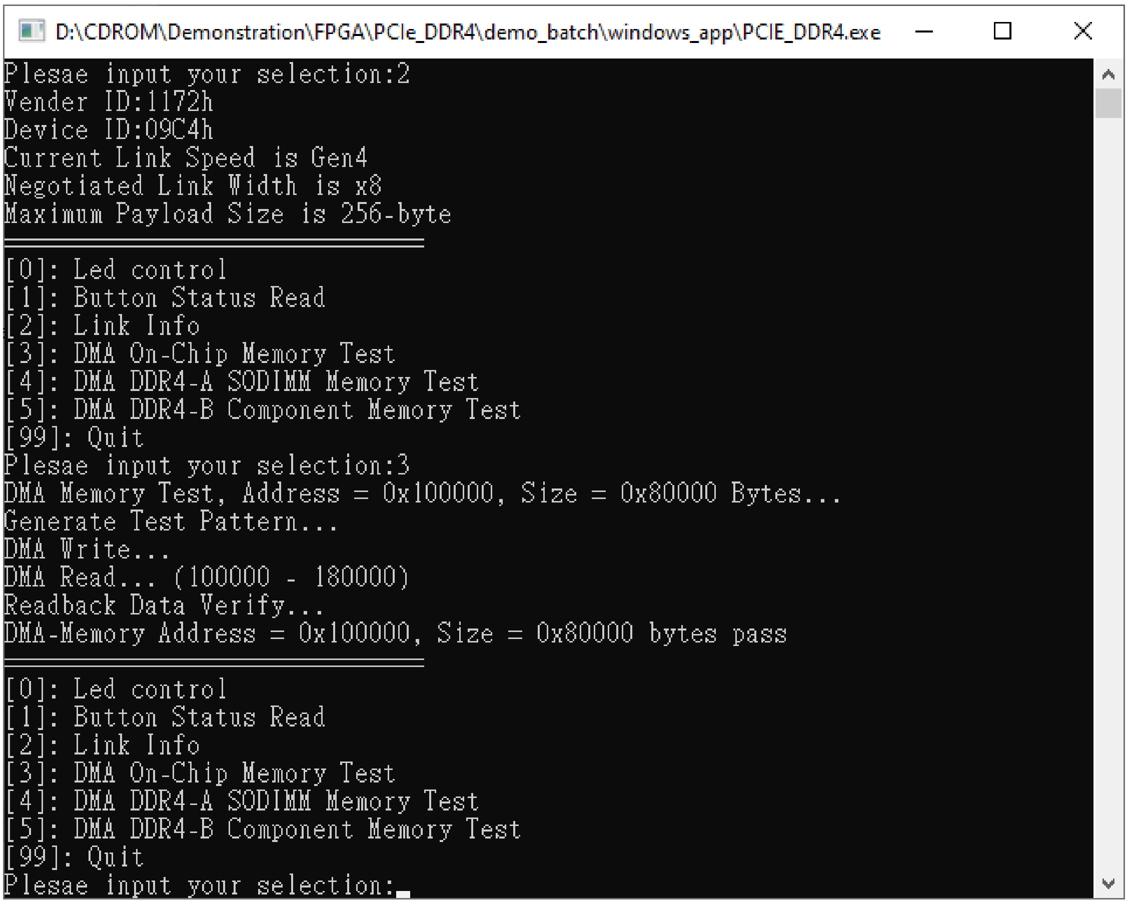


Figure 4-18 Screenshot of On-Chip Memory DMA Test Result

10. Type 4 followed by the ENTER key to select the DMA DDR4-A SODIMM Memory Test item. The DMA write and read test result will be reported as shown in **Figure 4-19**.

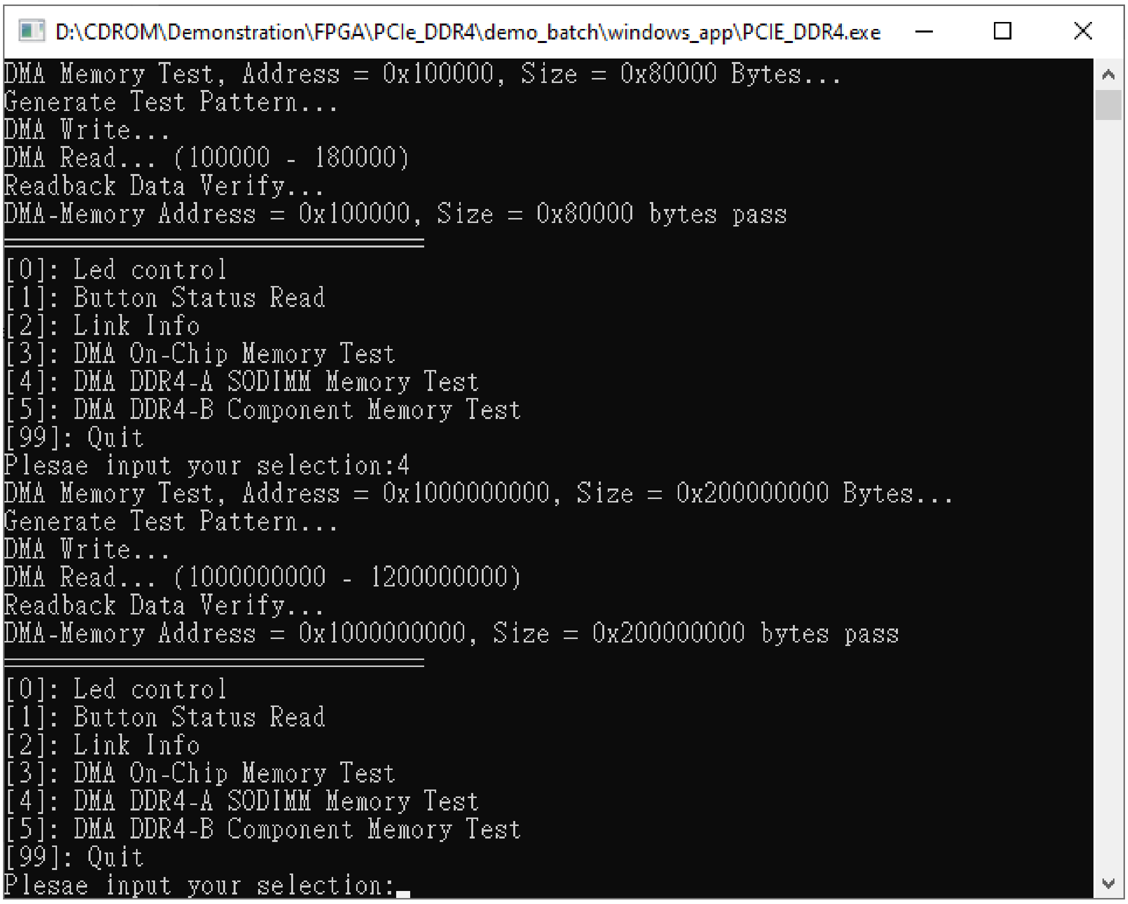


Figure 4-19 Screenshot of the DDR4-A SODIMM Memory DMA Test Result

11. Type 5 followed by the ENTER key to select the DMA DDR4-B Component Memory Test item. The DMA write and read test result will be reported as shown in **Figure 4-20**.

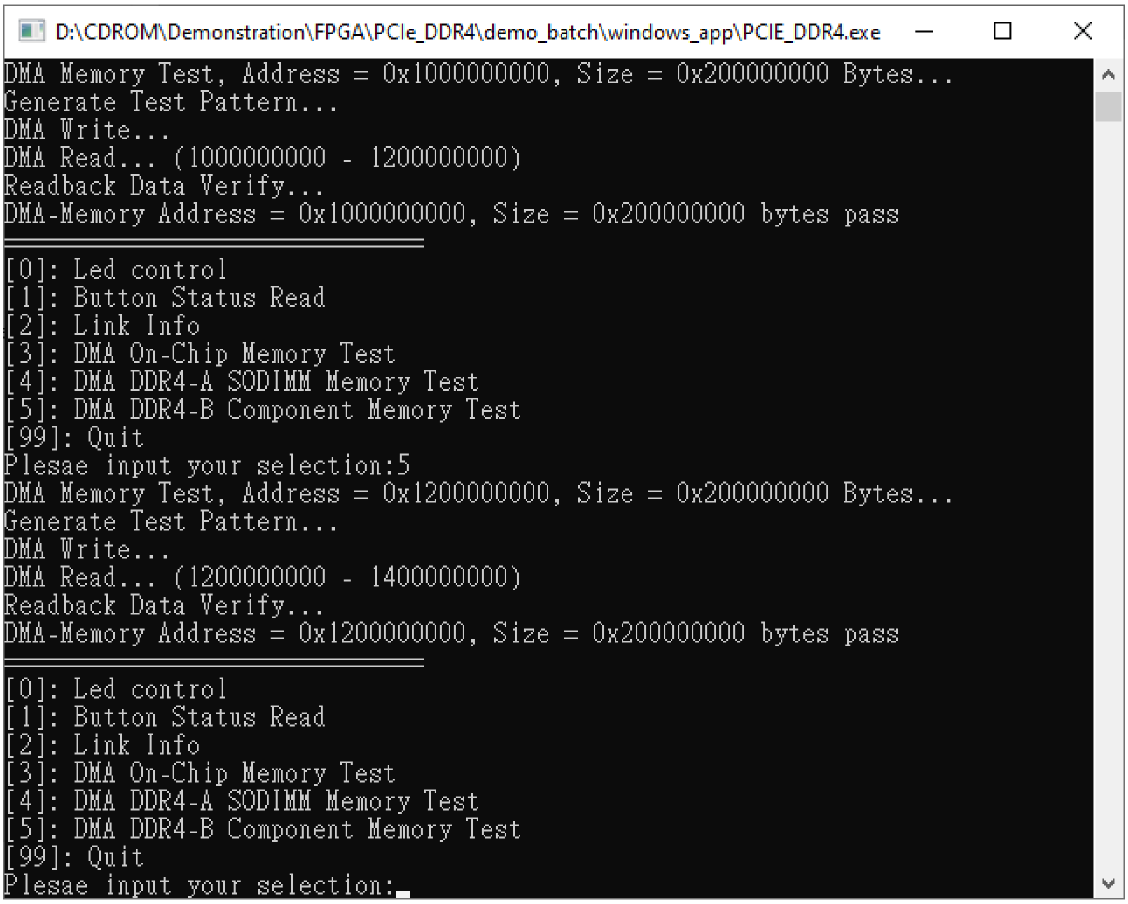


Figure 4-20 Screenshot of the DDR4-B Component Memory DMA Test Result

12. Type 99 followed by the ENTER key to exit this test program.

■ Development Tools

- Quartus Prime 23.2 Pro Edition
- Visual C++ 2019

■ Demonstration Source Code Location

- Quartus Project: Demonstrations\PCIE_DDR4
- Visual C++ Project: Demonstrations\PCIe_SW_KIT\Windows\PCIe_DDR4

■ FPGA Application Design

Figure 4-21 shows the system block diagram in the FPGA system. In the **Platform Designer** (formerly Qsys), the PIO controller is used to control the LED and monitor the Button Status, and the On-Chip memory is used for performing DMA testing. The PIO controllers and the On-Chip memory are connected to the PCI Express Hard IP controller through the Memory-Mapped Interface.

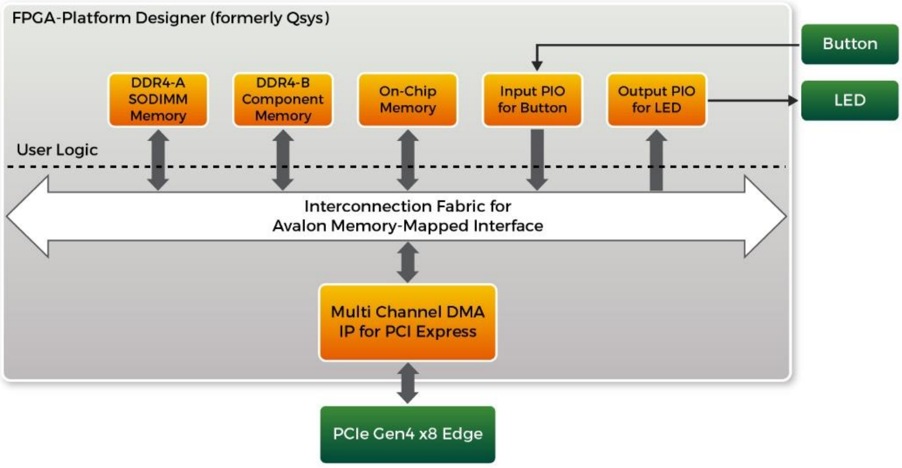


Figure 4-21 Hardware block diagram of the PCIe_DDR4 reference design

■ Windows Based Application Software Design

The application software project is built by Visual C++ 2019. The project includes the following major files:

Name	Description
PCIE_DDR4.cpp	Main program
PCIE.c	Implement dynamically load for TERASIC_PCIE_MCDMA.dll
PCIE.h	
TERASIC_PCIE_MCDMA.h	SDK library file, defines constant and data structure

The main program PCIE_DDR4.cpp includes the header file "PCIE.h" and defines the controller address according to the FPGA design.

```
#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x800000
#define DEMO_PCIE_IO_BUTTON_ADDR    0x800040
#define DEMO_PCIE_ONCHIP_MEM_ADDR   0x100000
#define DEMO_PCIE_DDR4A_MEM_ADDR    0x1000000000
#define DEMO_PCIE_DDR4B_MEM_ADDR    0x1200000000

#define ONCHIP_MEM_TEST_SIZE        (512*1024) //512KB
#define DDR4A_MEM_TEST_SIZE         (8ull*1024*1024*1024) //8GB
#define DDR4B_MEM_TEST_SIZE         (8ull*1024*1024*1024) //8GB
```

The base address of BUTTON and LED controllers are 0x800040 and 0x800000 based on PCIE_BAR4, respectively. The on-chip memory base address is 0x100000 relative to the DMA controller. The above definitions are the same as those in the PCIe Fundamental demo.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the TERASIC_PCIE_MCDMA.dll. Then, it calls PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in the PCIE_Open are defined in TERASIC_PCIE_MCDMA.h. If developers change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value defined in TERASIC_PCIE_MCDMA.h. If the return value of PCIE_Open is zero, it means the driver cannot be accessed successfully. In this case, please make sure:

- The FPGA is configured with the associated bit-stream file and the host is rebooted.
- The PCI express driver is loaded successfully.

The LED control is implemented by calling PCIE_Write32 API, as shown below:

```
bPass = PCIE_Write32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_LED_ADDR, (uint32_t) Mask);
```

The button status query is implemented by calling the **PCIE_Read32** API, as shown below:

```
bPass = PCIE_Read32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_BUTTON_ADDR, &Status);
```

The memory-mapped memory read and write test is implemented by **PCIE_DmaWrite** and **PCIE_DmaRead** API, as shown below:

```
bPass = PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
bPass = PCIE_DmaRead(hPCIe, LocalAddr, pRead, nTestSize);
```

The PCIe link information is implemented by PCIE_ConfigRead32 API, as shown below:

```
// read config - link status
if (PCIE_ConfigRead32(hPCIe, 0x80, &Data32)) {
    switch ((Data32 >> 16) & 0x0F) {
    case 1:
        printf("Current Link Speed is Gen1\n");
        break;
    case 2:
        printf("Current Link Speed is Gen2\n");
        break;
    case 3:
        printf("Current Link Speed is Gen3\n");
        break;
    case 4:
        printf("Current Link Speed is Gen4\n");
        break;
    default:
        printf("Current Link Speed is Unknown\n");
        break;
    }
    switch ((Data32 >> 20) & 0x3F) {
    case 1:
        printf("Negotiated Link Width is x1\n");
        break;
    case 2:
        printf("Negotiated Link Width is x2\n");
        break;
    case 4:
        printf("Negotiated Link Width is x4\n");
        break;
    case 8:
        printf("Negotiated Link Width is x8\n");
        break;
    case 16:
        printf("Negotiated Link Width is x16\n");
        break;
    default:
        printf("Negotiated Link Width is Unknown\n");
        break;
    }
} else {
    bPass = false;
}
```

Chapter 5

PCI Express Reference Design for Linux

PCI Express is commonly used in consumer, server, and industrial applications, to link motherboard-mounted peripherals. From this demonstration, it will show how the PC Linux and FPGA communicate with each other through the PCI Express interface. Multi Channel DMA Intel® FPGA IP for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel document ug20297-683821-750934.

5.1 PCI Express System Infrastructure

Figure 5-1 shows the infrastructure of the PCI Express System in this demonstration. It consists of two primary components: FPGA System and PC System. The FPGA System is developed based on Multi Channel DMA Intel® FPGA IP for PCI Express. The application software on the PC side is developed by Terasic based on Intel's PCIe kernel mode driver.

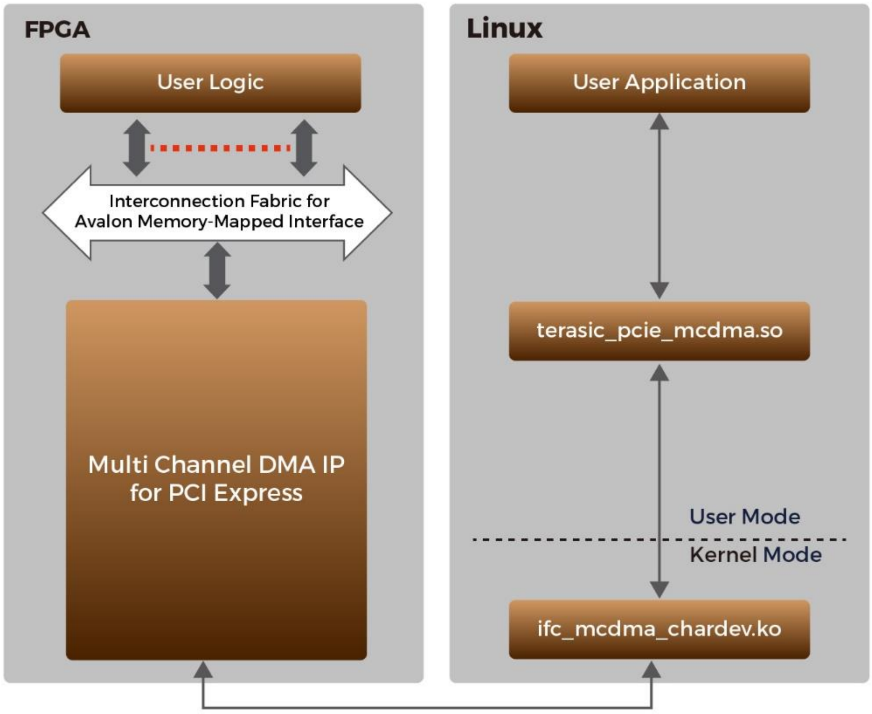


Figure 5-1 Infrastructure of PCI Express System

5.2 PC PCI Express Software SDK

The FPGA System CD contains a PC Linux based SDK to allow users to develop their 64-bit software application on 64-bits Linux. Ubuntu 20.04 is recommended. The SDK is located in the "CDROM/Demonstrations/PCIe_SW_KIT/Linux" folder which includes:

- PCI Express Driver
- PCI Express Library
- PCI Express Examples

The kernel mode driver assumes the PCIe vendor ID (VID) is 0x1172 and the device ID (DID) is 0x09C4. If different VID and DID are used in the design, users need to modify the PCIe vendor ID (VID) and device ID (DID) in the driver config_file file accordingly.

The PCI Express Library is implemented as a single .so file named terasic_pcie_mcdma.so.
This file is a 64-bit library file. With the library exported software API, users can easily communicate with the FPGA. The library provides the following functions:

- Basic data read and write
- Data read and write by DMA

For high performance data transmission, MCDMA is required as the read and write operations are specified under the hardware design on the FPGA.

5.3 PCI Express Software Stack

Figure 5-2 shows the software stack for the PCI Express application software on 64-bit Linux. The PCIe library module terasic_pcie_mcdma.so provides DMA and direct I/O access for user application program to communicate with FPGA. Users can develop their applications based on this .so library file. The ifc_mcdma_chardev.ko kernel driver is provided by Intel.

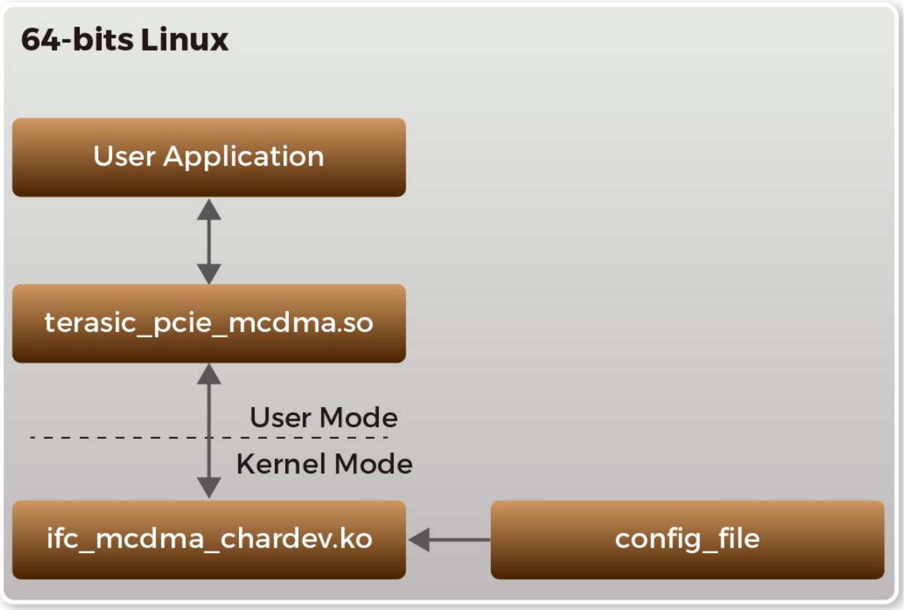


Figure 5-2 PCI Express Software Stack

■ Install PCI Express Driver on Linux

To make sure the PCIe driver can meet your kernel of Linux distribution, the driver ifc_mcdma_chardev.ko should be recompiled before it is used. The PCIe driver project is locate in the folder:

"CDROM/Demonstrations/PCIe_SW_KIT/Linux/PCIe_Driver"

The folder includes the following files:

- ifc_mcdma_chr.c

- ifc_mcdma_chr.h
- ifc_mcdma_pci.c
- ifc_mcdma_pci.h
- common/include/regs/pio_reg_registers.h
- common/include/regs/qdma_regs_2_registers.h
- common/include/ifc_mcdma.h
- common/include/ifc_mcdma_utils.h
- common/include/mcdma_ip_params.h
- common/mk/common.mk
- common/mk/env.mk
- common/src/ifc_mcdma_utils.c
- Makefile
- load_driver
- unload
- config_file

To compile and install the PCI Express driver, please execute the steps below:

1. Install the board the PCIe slot of the host PC
2. Make sure Quartus Programmer and USB-Blaster II driver are installed
3. Open a terminal and use "cd" command to go to the folder "CDROM/Demonstrations/PCIe_Fundamental/demo_batch".
4. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace "/home/user/intelFPGA_pro/23.2/quartus" to your quartus installation path.

   ```
   export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/23.2/quartus
   ```
5. Execute "sudo -E sh test.sh" command to configure the FPGA
6. Restart the Linux operation system. In Linux, open a terminal and use "cd" command to goto the PCIe_Driver folder
7. Type the following commands to compile and install the driver ifc_mcdma_chardev.ko, and make sure driver is loaded successfully and FPGA is detected by the driver as shown in **Figure 5-3.**
   - make
   - sudo sh load_driver
   - dmesg | tail -n 10

```
user@user-Z590-VISION-G:PCIe_Driver$ dmesg | tail -n 10
[    9.767941] input: HDA Intel PCH HDMI/DP,pcm=17 as /devices/pci0000:00/0000:00:1f.3/sound/card0/input25
[   10.292848] usbcore: registered new interface driver snd-usb-audio
[   10.653291] rfkill: input handler disabled
[   12.082977] igc 0000:04:00.0 enp4s0: NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX/TX
[   12.083236] IPv6: ADDRCONF(NETDEV_CHANGE): enp4s0: link becomes ready
[   84.927087] ifc_mcdma_chardev: loading out-of-tree module taints kernel.
[   84.927115] ifc_mcdma_chardev: module verification failed: signature and/or required key missing - tainting kernel
[   84.927664] ifc_mcdma Intel(R) PCIe end point driver - version 0.5.0.0
[   84.927668] Copyright (c) 2020-21, Intel Corporation.
[   84.927740] ifc_mcdma_chardev 0000:01:00.0: enabling device (0000 -> 0002)
user@user-Z590-VISION-G:PCIe_Driver$
```

Figure 5-3 Screenshot of install PCIe driver

■ Create a Software Application

All the files needed to create a PCIe software application are located in the directory CDROM/Demonstrations/PCIe_SW_KIT/Linux/PCIe_Library. It includes the following files:

- TERASIC_PCIE_MCDMA.h
- terasic_pcie_mcdma.so (64-bit library)

Below lists the procedures to use the library in users' C/C++ project:

1. Create a 64-bit C/C++ project.
2. Include TERASIC_PCIE_MCDMA.h in the C/C++ project.
3. Copy terasic_pcie_mcdma.so to the folder where the project execution file is located.
4. Link terasic_pcie_mcdma.so in C/C++ program. To load the terasic_pcie_mcdma.so, please refer to the PCIe fundamental example below.
5. Call the library API to implement the desired application.

Users can easily communicate with the FPGA through the PCIe bus through the terasic_pcie_mcdma.so API. The details of API are described below sections.

5.4 PCI Express Library API

The API is the same as Windows Library. Please refer to the section **PCI Express Library API** in this document.

5.5 PCIe Reference Design - Fundamental

The application reference design shows how to implement fundamental control and data transfer in the DMA. In the design, basic I/O is used to control the BUTTON and LED on the FPGA board. High-speed data transfer is performed by the DMA.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM/Demonstrations/PCIe_Fundamental/demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_Fundamental.sof
- Download Batch file: test.sh
- Linux Application Software folder : linux_app, includes
  - ✧ PCIE_FUNDAMENTAL
  - ✧ terasic_pcie_mcdma.so

■ Demonstration Setup

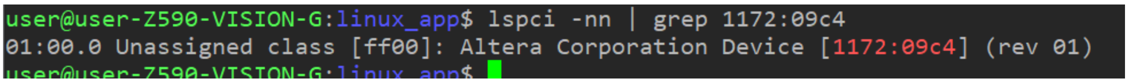
1. Install the FPGA board on your PC as shown in **Figure 5-4**.

Figure 5-4 FPGA board installation on PC

2. Open a terminal and use "cd" command to goto "CDROM/Demonstrations/PCIe_Fundamental/demo_batch".
3. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace /home/user/intelFPGA_pro/23.2/quartus to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/23.2/quartus
```

4. Execute "sudo -E sh test.sh" command to configure the FPGA
5. Restart Linux
6. Install PCIe driver. The driver is located in the folder:
   CDROM/Demonstration/PCIe_SW_KIT/Linux/PCIe_Driver.
7. Type “lspci -nn | grep 1172:09c4” to make sure the Linux has detected the FPGA Board as shown below.



8. Goto linux_app folder, execute "sudo ./PCIE_FUNDAMENTAL". A menu will appear as shown in **Figure 5-5**.

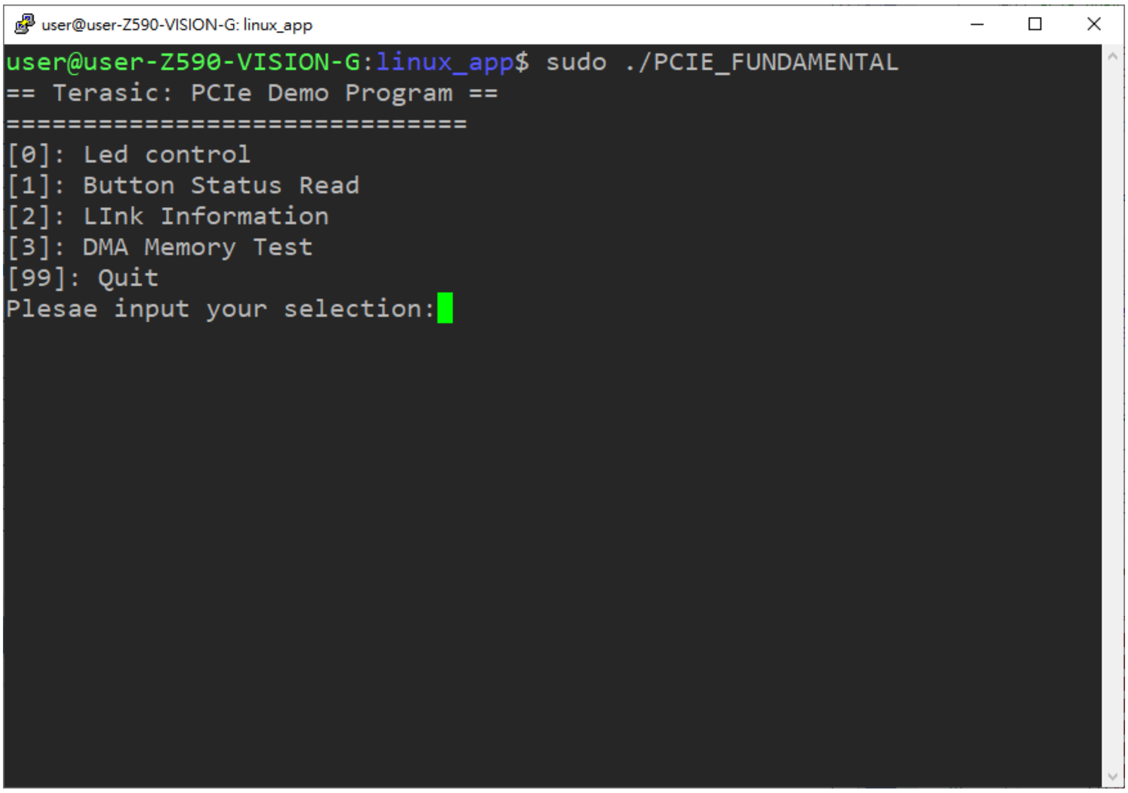
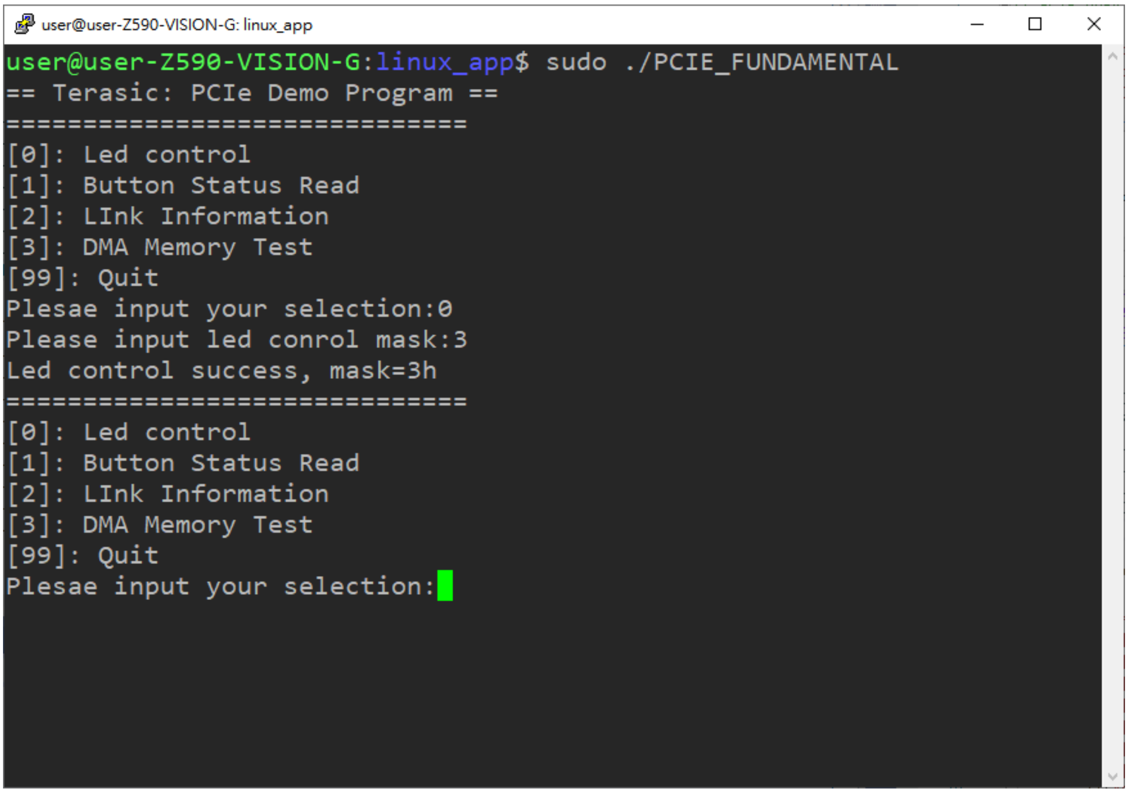


Figure 5-5 Screenshot of Program Menu

9. Type 0 followed by the ENTER key to select the Led Control item, then input 3 (hex 0x03) will turn all leds on as shown in **Figure 5-6**. If input 0 (hex 0x00), all led will be turned off.



```
user@user-Z590-VISION-G: linux_app
user@user-Z590-VISION-G:linux_app$ sudo ./PCIE_FUNDAMENTAL
== Terasic: PCIe Demo Program ==
==============================
[0]: Led control
[1]: Button Status Read
[2]: LInk Information
[3]: DMA Memory Test
[99]: Quit
Plesae input your selection:0
Please input led conrol mask:3
Led control success, mask=3h
==============================
[0]: Led control
[1]: Button Status Read
[2]: LInk Information
[3]: DMA Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 5-6 Screenshot of LED Control

10. Type 1 followed by the ENTER key to select the Button Status Read item. The button status will be reported as shown in **Figure 5-7**.

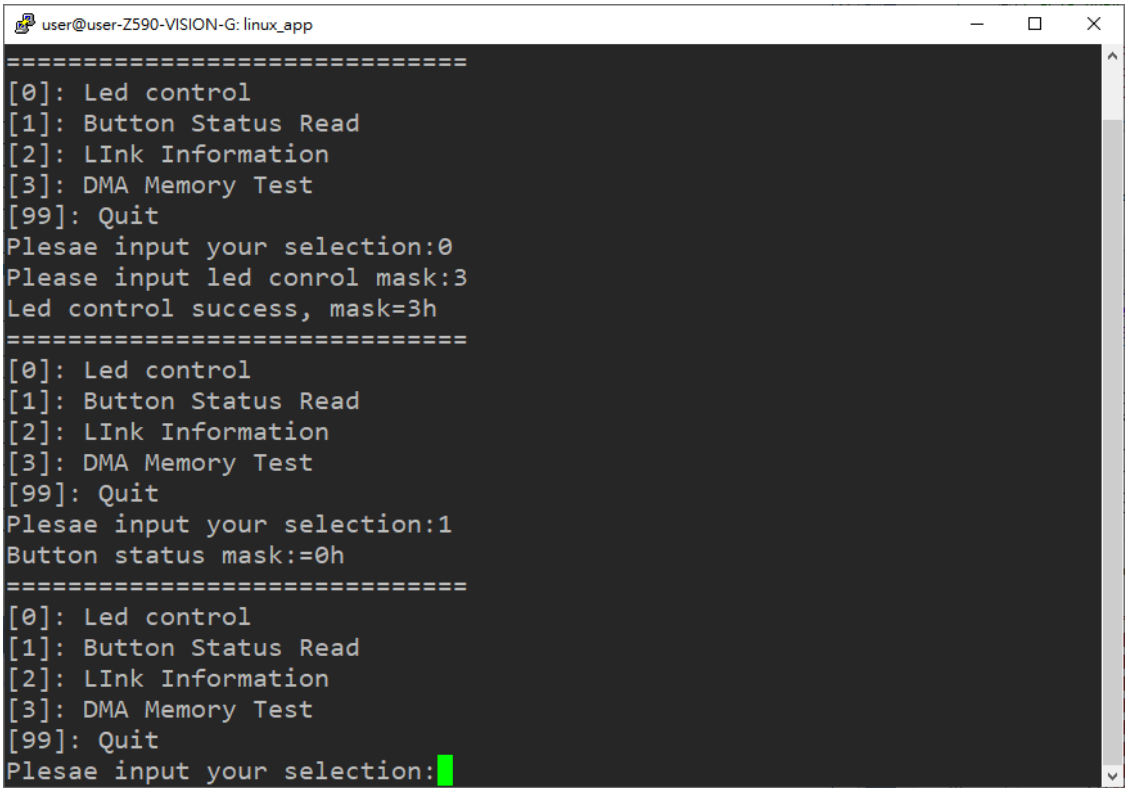


Figure 5-7 Screenshot of Button Status Report

11. Type 3 followed by the ENTER key to select the DMA Testing item. The DMA test result will be reported as shown in **Figure 5-8**.

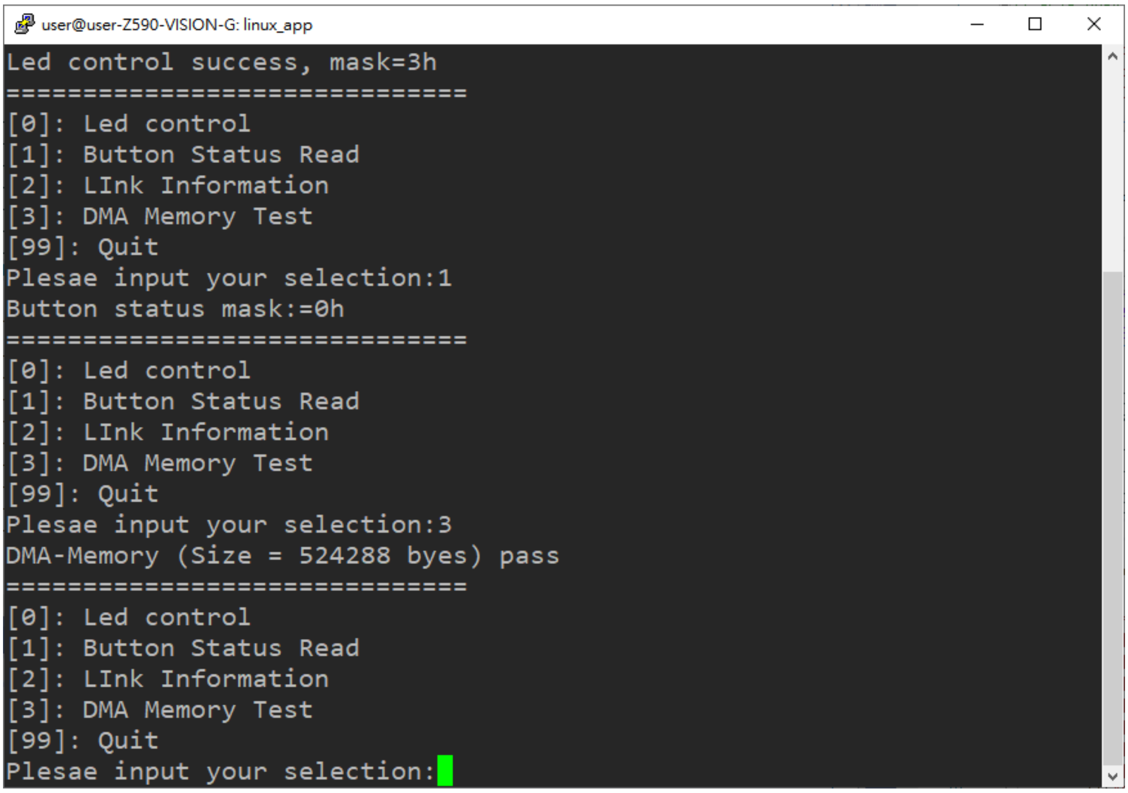


Figure 5-8 Screenshot of DMA Memory Test Result

12. Type 99 followed by the ENTER key to exit this test program

■ Development Tools

- Quartus Prime 23.2 Pro Edition
- GNU Compiler Collection, Version 9.4 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstrations/PCIe_Fundamental
- C++ Project: Demonstrations/PCIe_SW_KIT/Linux/PCIE_FUNDAMENTAL

■ FPGA Application Design

Figure 5-9 shows the system block diagram in the FPGA system. In the **Platform Designer** (formerly Qsys), the PIO controller is used to control the LED and monitor the Button Status, and the On-Chip memory is used for performing DMA testing. The PIO controllers and the On-Chip memory are connected to the PCI Express Hard IP controller through the Memory-Mapped Interface.

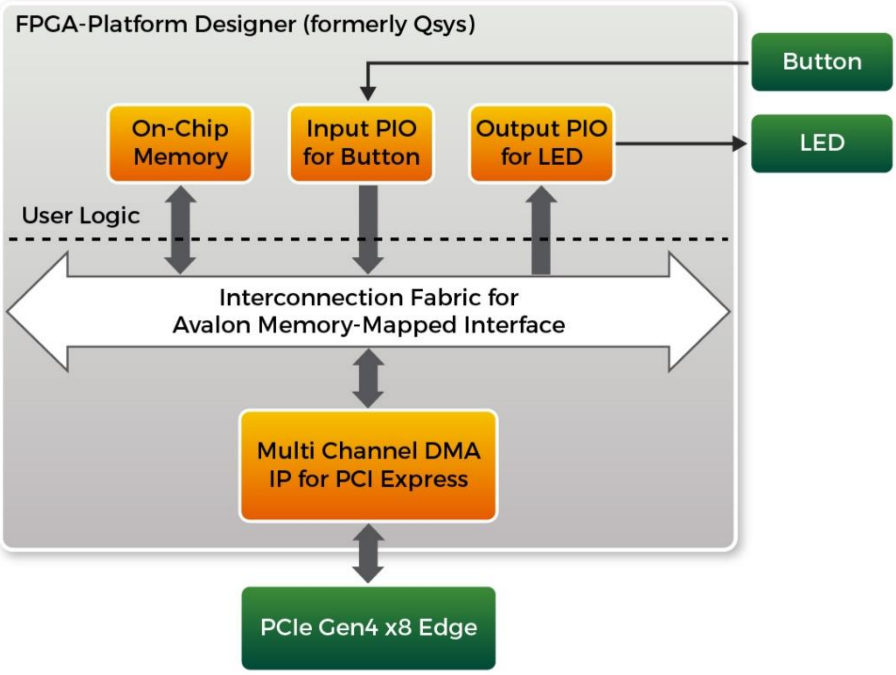


Figure 5-9 Hardware block diagram of the PCIe reference design

■ Linux Based Application Software Design

The application software project is built by GNU Toolchain. The project includes the following major files:

Name	Description
PCIE_FUNDAMENTAL.cpp	Main program
TERASIC_PCIE_MCDMA.h	SDK library file, defines constant and data structure

The main program PCIE_FUNDAMENTAL.cpp defines the controller address according to the FPGA design.

```
#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x800000
#define DEMO_PCIE_IO_BUTTON_ADDR    0x800040
#define DEMO_PCIE_MEM_ADDR          0x100000

#define MEM_SIZE                    (512*1024) //512KB
```

The base address of BUTTON and LED controllers are 0x800040 and 0x800000 based on PCIE_BAR4, respectively. The on-chip memory base address is 0x100000 relative to the DMA controller.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the terasic_pcie_mcdma.so. Then, it call PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in PCIE_Open are defined in TERASIC_PCIE_MCDMA.h. If developers change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value defined in TERASIC_PCIE_MCDMA.h. If the return value of PCIE_Open is zero, it means the driver cannot be accessed successfully. In this case, please make sure:

- The FPGA is configured with the associated bit-stream file and the host is rebooted.
- The PCI express driver is loaded successfully.

The LED control is implemented by calling PCIE_Write32 API, as shown below:

```
bPass = PCIE_Write32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_LED_ADDR, (uint32_t) Mask);
```

The button status query is implemented by calling the **PCIE_Read32** API, as shown below:

```
bPass = PCIE_Read32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_BUTTON_ADDR, &Status);
```

The memory-mapped memory read and write test is implemented by **PCIE_DmaWrite** and **PCIE_DmaRead** API, as shown below:

```
bPass = PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
bPass = PCIE_DmaRead(hPCIe, LocalAddr, pRead, nTestSize);
```

5.6 PCIe Reference Design - DDR4

The application reference design shows how to add DDR4 Memory Controllers for DDR4-A SODIMM and DDR4-B Component into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 8GB data DMA for both SODIMM. Also, this demo shows how to call "PCIE_ConfigRead32" API to check PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_DDR4\demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_DDR4.sof
- Download Batch file: test.sh
- Linux Application Software folder : linux_app, includes
    - ✧ PCIE_DDR4
    - ✧ terasic_pcie_mcdma.so

■ Demonstration Setup

1. Install one pieces of DDR4 8GB SODIMM on the FPGA board.
2. Install the FPGA board on the PCIe Slot of your PC.
3. Open a terminal and use "cd" command to go to "CDROM/Demonstrations/PCIe_DDR4/demo_batch".
4. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in the terminal. Replace /home/user/intelFPGA_pro/23.2/quartus to your Quartus installation path.

```
export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/23.2/quartus
```

5. Execute "sudo -E sh test.sh" command to configure the FPGA
6. Restart Linux
7. Install PCIe driver.
8. Make sure that Linux has detected the FPGA Board.
9. Go to the linux_app folder, execute "sudo ./PCIE_DDR4". A menu will appear as shown in **Figure 5-10**.

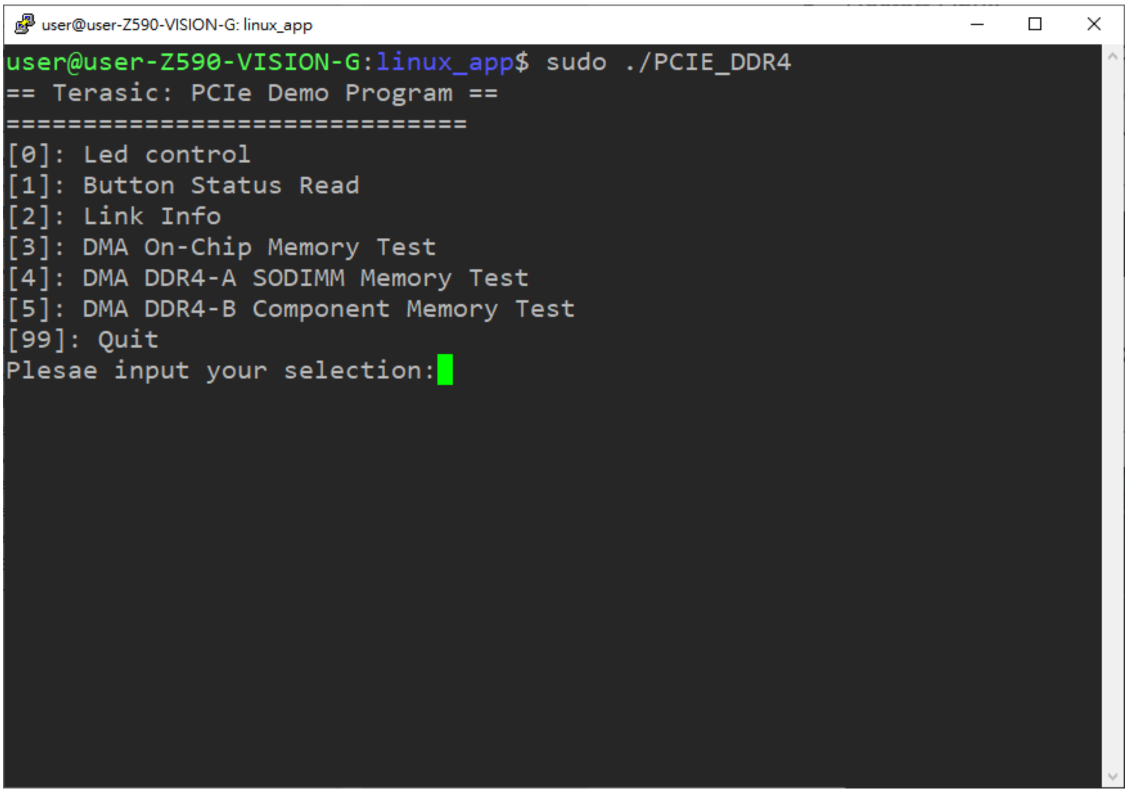


Figure 5-10 Screenshot of Program Menu

10. Type 2 followed by the ENTER key to select the Link Info item. The PCIe link information will be shown as in **Figure 5-11**. Gen4 link speed and x8 link width are expected.

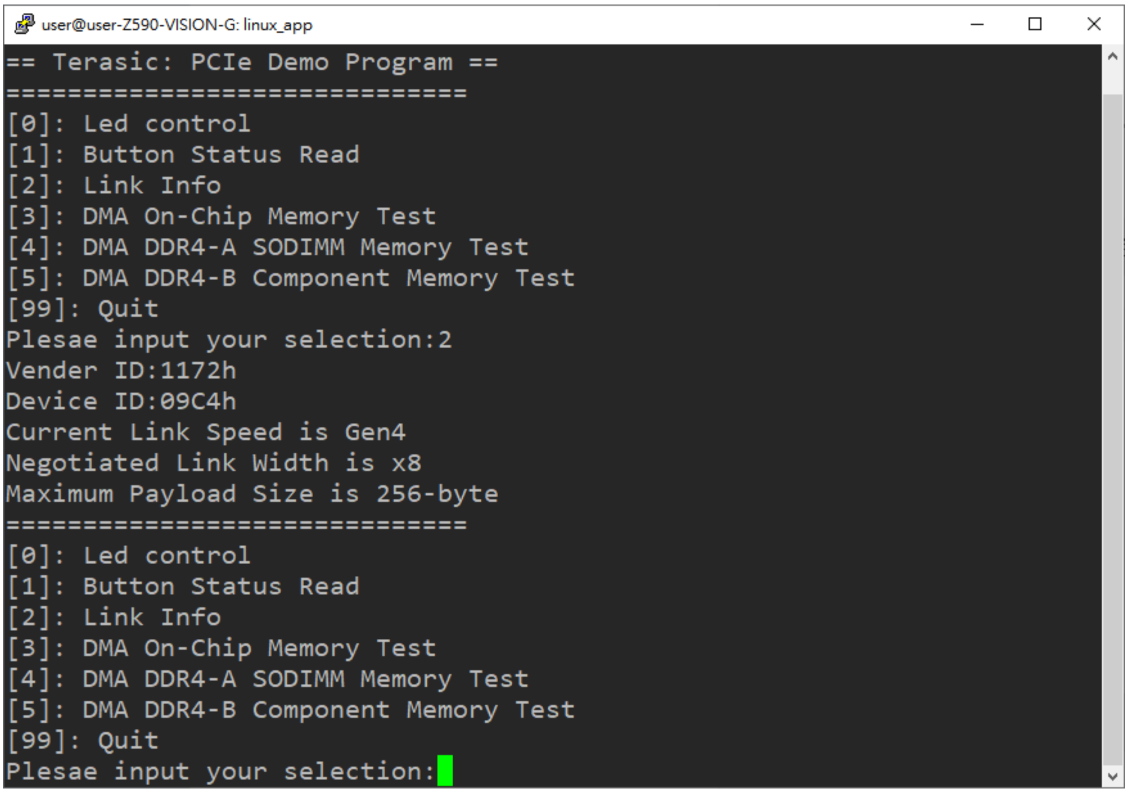


Figure 5-11 Screenshot of Link Info

11. Type 3 followed by the ENTER key to select DMA On-Chip Memory Test item. The DMA write and read test result will be report as shown in **Figure 5-12**.

```
user@user-Z590-VISION-G: linux_app
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A SODIMM Memory Test
[5]: DMA DDR4-B Component Memory Test
[99]: Quit
Plesae input your selection:3
DMA Memory Test, Address = 0x100000, Size = 0x80000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read... (100000 - 180000)
Readback Data Verify...
DMA-Memory Address = 0x100000, Size = 0x80000 bytes pass
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A SODIMM Memory Test
[5]: DMA DDR4-B Component Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 5-12 Screenshot of On-Chip Memory DMA Test Result

12. Type 4 followed by the ENTER key to select the DMA DDR4-A SODIMM Memory Test item. The DMA write and read test result will be reported as shown in **Figure 5-13**.

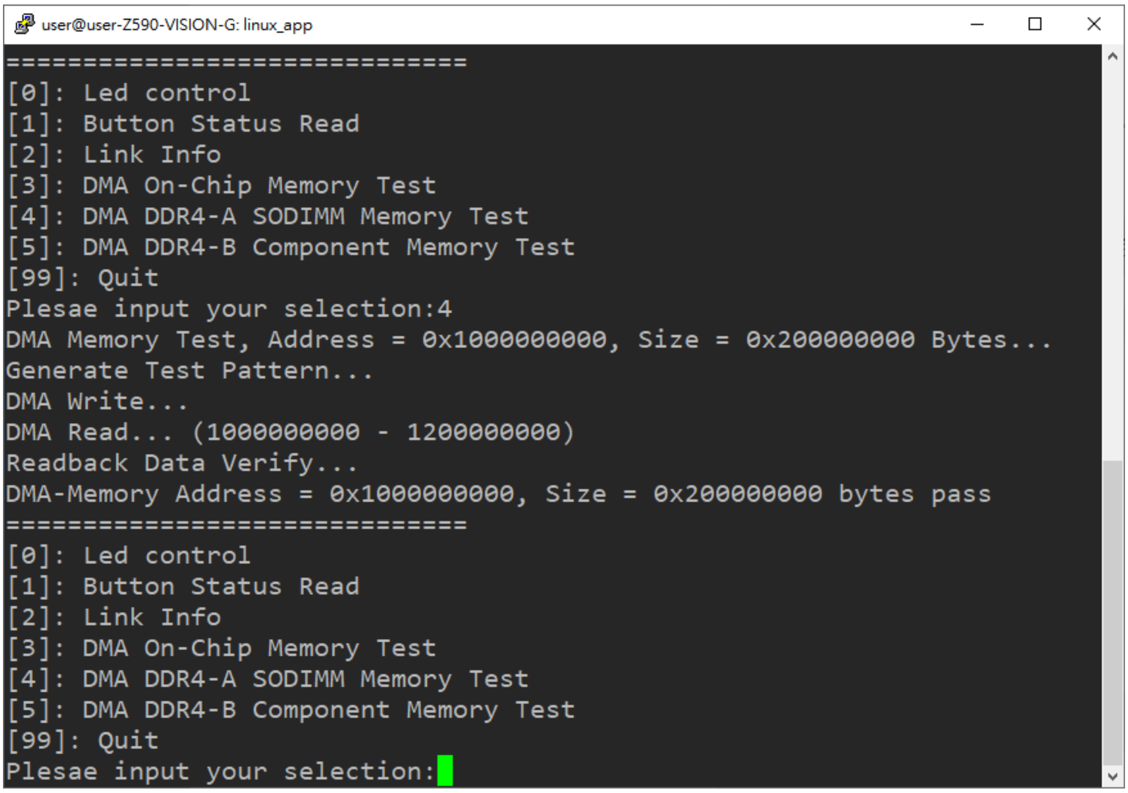


Figure 5-13 Screenshot of DDR4-A SOIMM Memory DAM Test Result

13. Type 5 followed by the ENTER key to select the DMA DDR4-B Component Memory Test item. The DMA write and read test result will be reported as shown in **Figure 5-14**.

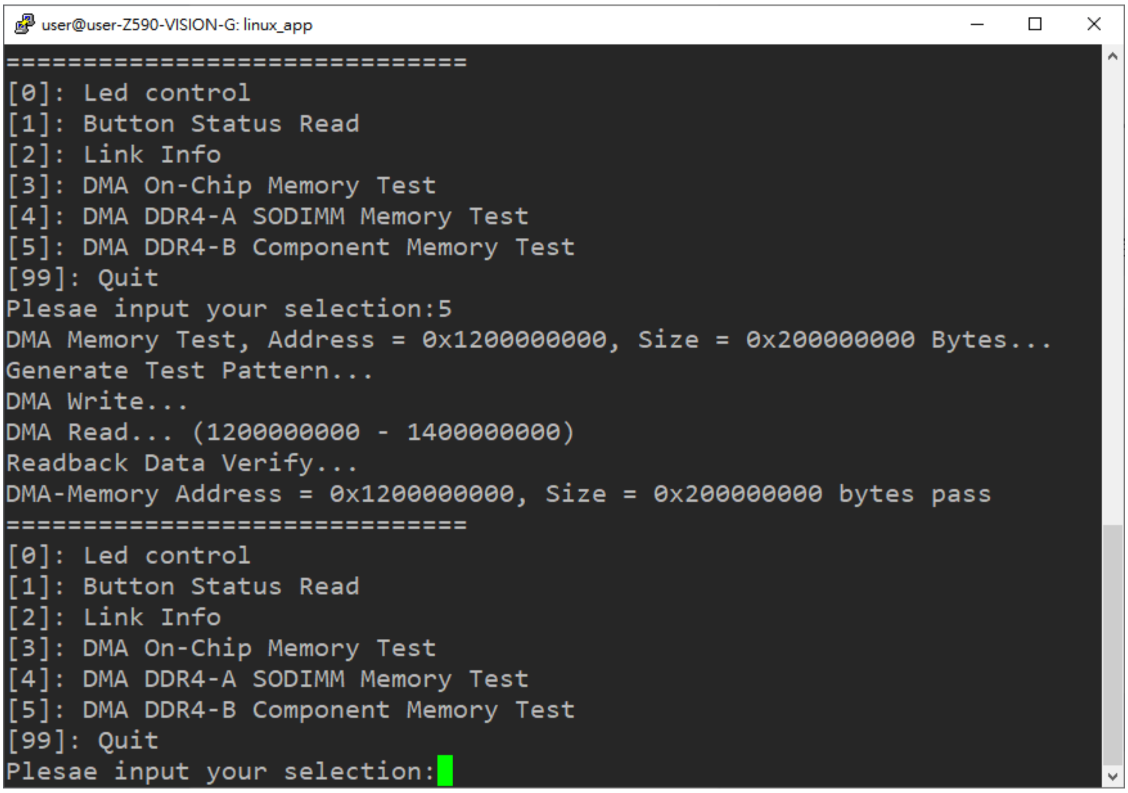


Figure 5-14 Screenshot of DDR4-B Component Memory DAM Test Result

14. Type 99 followed by the ENTER key to exit this test program.

■ Development Tools

- Quartus Prime 23.2 Pro Edition
- GNU Compiler Collection, Version 9.4 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstrations/PCIE_DDR4
- C++ Project: Demonstrations/PCIe_SW_KIT/Linux/PCIe_DDR4

■ FPGA Application Design

Figure 5-15 shows the system block diagram in the FPGA system. In the **Platform Designer** (formerly Qsys), the PIO controller is used to control the LED and monitor the Button Status, and the On-Chip memory is used for performing DMA testing. The PIO controllers and the On-Chip memory are connected to the PCI Express Hard IP controller through the Memory-Mapped Interface.

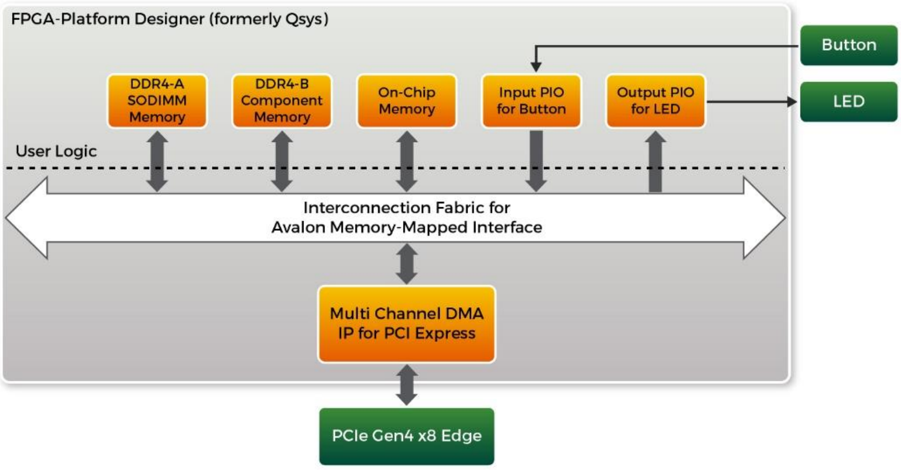


Figure 5-15 Hardware block diagram of the PCIe_DDR4 reference design

■ Linux Based Application Software Design

The application software project is built by GNU Toolchain. The project includes the following major files:

Name	Description
PCIE_DDR4.cpp	Main program
TERASIC_PCIE_MCDMA.h	SDK library file, defines constant and data structure

The main program PCIE_DDR4.cpp defines the controller address according to the FPGA design.

```
#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x800000
#define DEMO_PCIE_IO_BUTTON_ADDR    0x800040
#define DEMO_PCIE_ONCHIP_MEM_ADDR   0x100000
#define DEMO_PCIE_DDR4A_MEM_ADDR    0x1000000000
#define DEMO_PCIE_DDR4B_MEM_ADDR    0x1200000000

#define ONCHIP_MEM_TEST_SIZE        (512*1024) //512KB
#define DDR4A_MEM_TEST_SIZE         (8ull*1024*1024*1024) //8GB
#define DDR4B_MEM_TEST_SIZE         (8ull*1024*1024*1024) //8GB
```

The base address of BUTTON and LED controllers are 0x800040 and 0x800000 based on PCIE_BAR4, respectively. The on-chip memory base address is 0x100000 relative to the DMA controller. The above definition is the same as those in PCIe Fundamental demo.

Before accessing the FPGA through PCI Express, the application first calls the PCIE_Load to dynamically load the terasic_pcie_mcdma.so. Then, it calls the PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in the PCIE_Open are defined in TERASIC_PCIE_MCDMA.h. If developers changes the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value defined in TERASIC_PCIE_MCDMA.h. If the return value of the PCIE_Open is zero, it means the driver cannot be accessed successfully. In this case, please make sure:

- The FPGA is configured with the associated bit-stream file and the host is rebooted.
- The PCI express driver is loaded successfully.

The LED control is implemented by calling PCIE_Write32 API, as shown below:

```
bPass = PCIE_Write32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_LED_ADDR, (uint32_t) Mask);
```

The button status query is implemented by calling the **PCIE_Read32** API, as shown below:

```
bPass = PCIE_Read32(hPCIe, DEMO_PCIE_USER_BAR, DEMO_PCIE_IO_BUTTON_ADDR, &Status);
```

The memory-mapped memory read and write test is implemented via **PCIE_DmaWrite** and the **PCIE_DmaRead** API, as shown below:

```
bPass = PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
bPass = PCIE_DmaRead(hPCIe, LocalAddr, pRead, nTestSize);
```

The PCIe link information is implemented by PCIE_ConfigRead32 API, as shown below:

```
// read config - link status
if (PCIE_ConfigRead32(hPCIe, 0x80, &Data32)) {
    switch ((Data32 >> 16) & 0x0F) {
    case 1:
        printf("Current Link Speed is Gen1\n");
        break;
    case 2:
        printf("Current Link Speed is Gen2\n");
        break;
    case 3:
        printf("Current Link Speed is Gen3\n");
        break;
    case 4:
        printf("Current Link Speed is Gen4\n");
        break;
    default:
        printf("Current Link Speed is Unknown\n");
        break;
    }
    switch ((Data32 >> 20) & 0x3F) {
    case 1:
        printf("Negotiated Link Width is x1\n");
        break;
    case 2:
        printf("Negotiated Link Width is x2\n");
        break;
    case 4:
        printf("Negotiated Link Width is x4\n");
        break;
    case 8:
        printf("Negotiated Link Width is x8\n");
        break;
    case 16:
        printf("Negotiated Link Width is x16\n");
        break;
    default:
        printf("Negotiated Link Width is Unknown\n");
        break;
    }
} else {
    bPass = false;
}
```

Chapter 6

Transceiver Verification

This chapter describes how to quickly verify the FPGA transceivers via the QSFP28 connector.

6.1 Transceiver Test Code

The transceiver test code is used to verify the transceiver channels via the QSPF28 ports through an external loopback method. The transceiver channels are verified with the data rates 25.78125 Gbps with **NRZ** modulation.

6.2 Loopback Fixture

To enable an external loopback of the transceiver channels, QSFP28 loopback fixtures, as shown in **Figure 9-1**, are required. The fixture is available at:

https://multilaneinc.com/product/ml4002-28/

Figure 6-1 QSFP28 Loopback Cable

Figure 6-2 shows the FPGA board with four QSFP28 loopback fixtures installed.

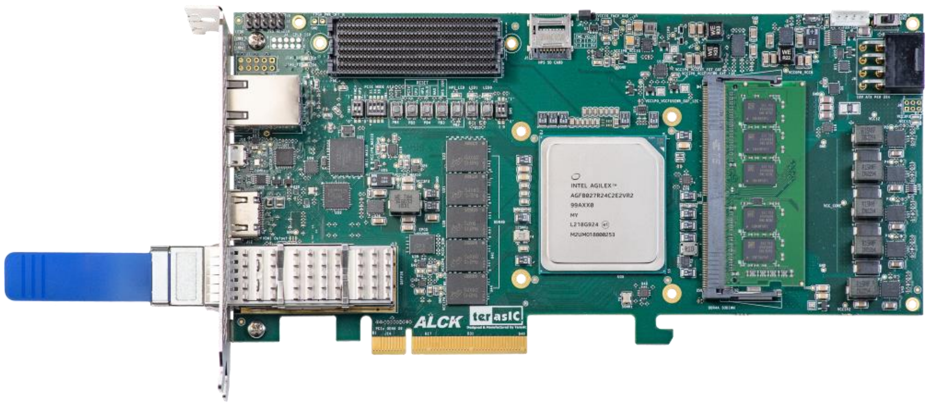


Figure 6-2 QSFP28 Transceiver Loopback Test in Progress

6.3 Testing by Transceiver Test Code

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

Figure 9 3 and Figure 9 4 shows the F-Tile PMA/FEC Direct PHY settings in the test code. The data rate of each transceiver channel is set to 25781.25 Mbps and the PMA modulation type is NRZ. So the 100Gbps QSPF28 loopback test code is implemented (4 channels in total). Also, the F-title Reference and System PLL setting is shown in Figure x.

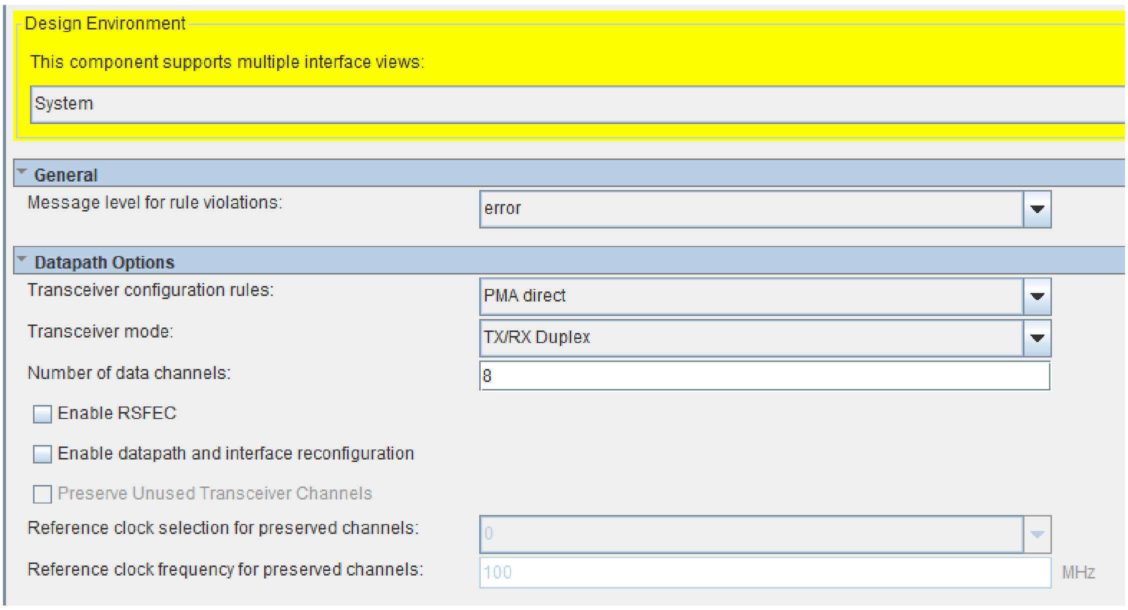


Figure 6-3 The Transceiver PHY setting

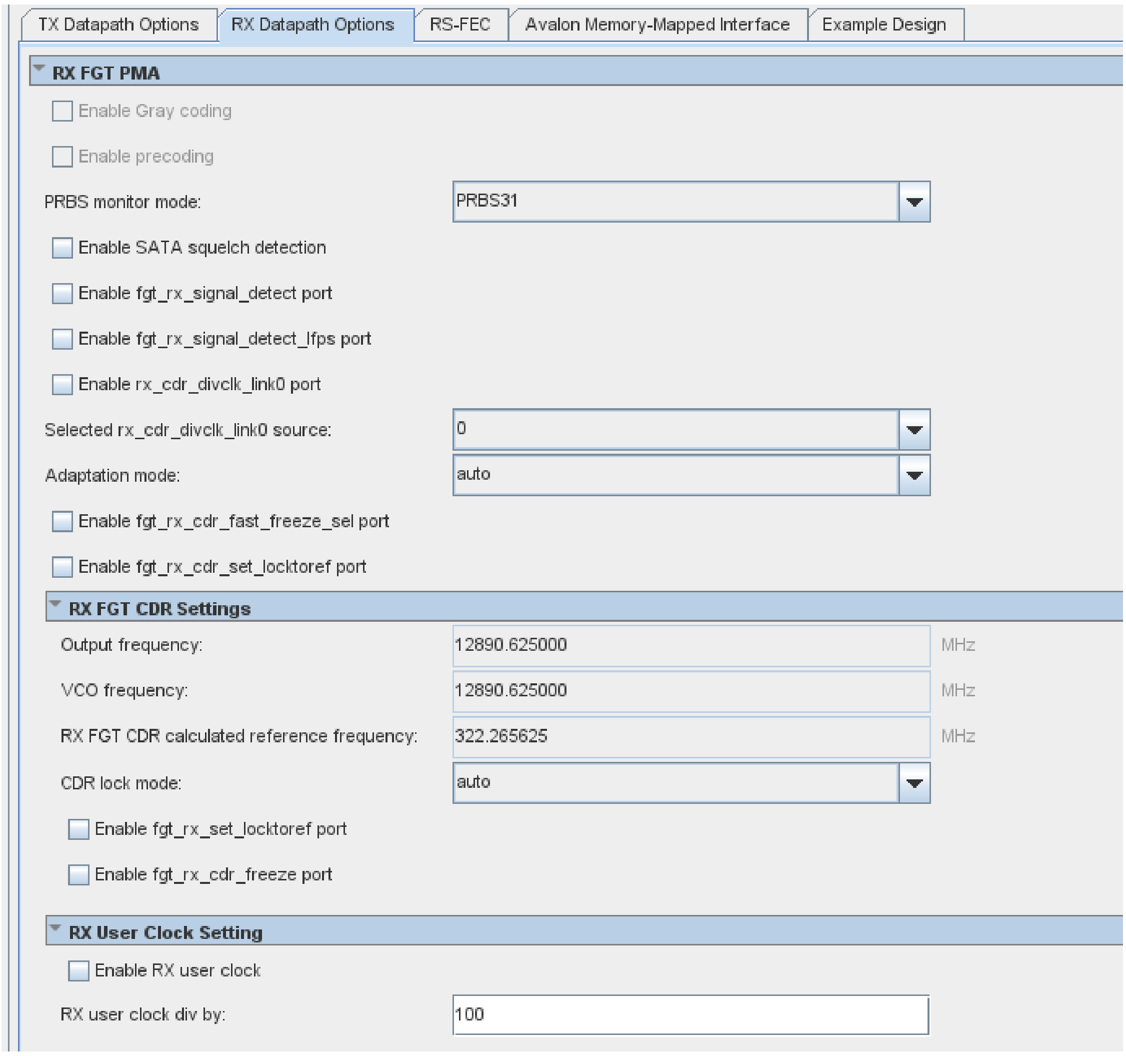


Figure 6-4 The Transceiver PHY setting

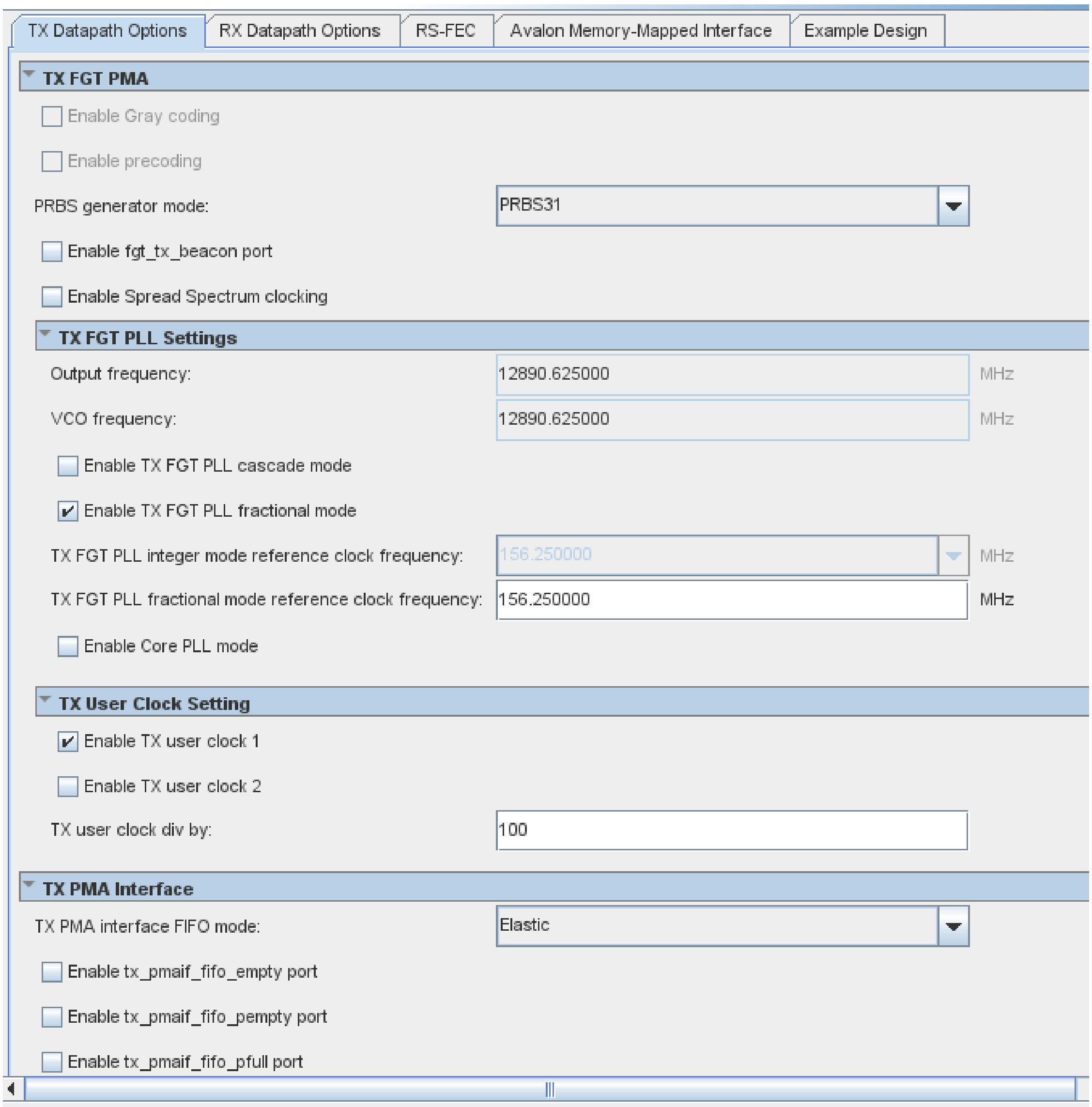


Figure 6-5 The Transceiver PHY setting

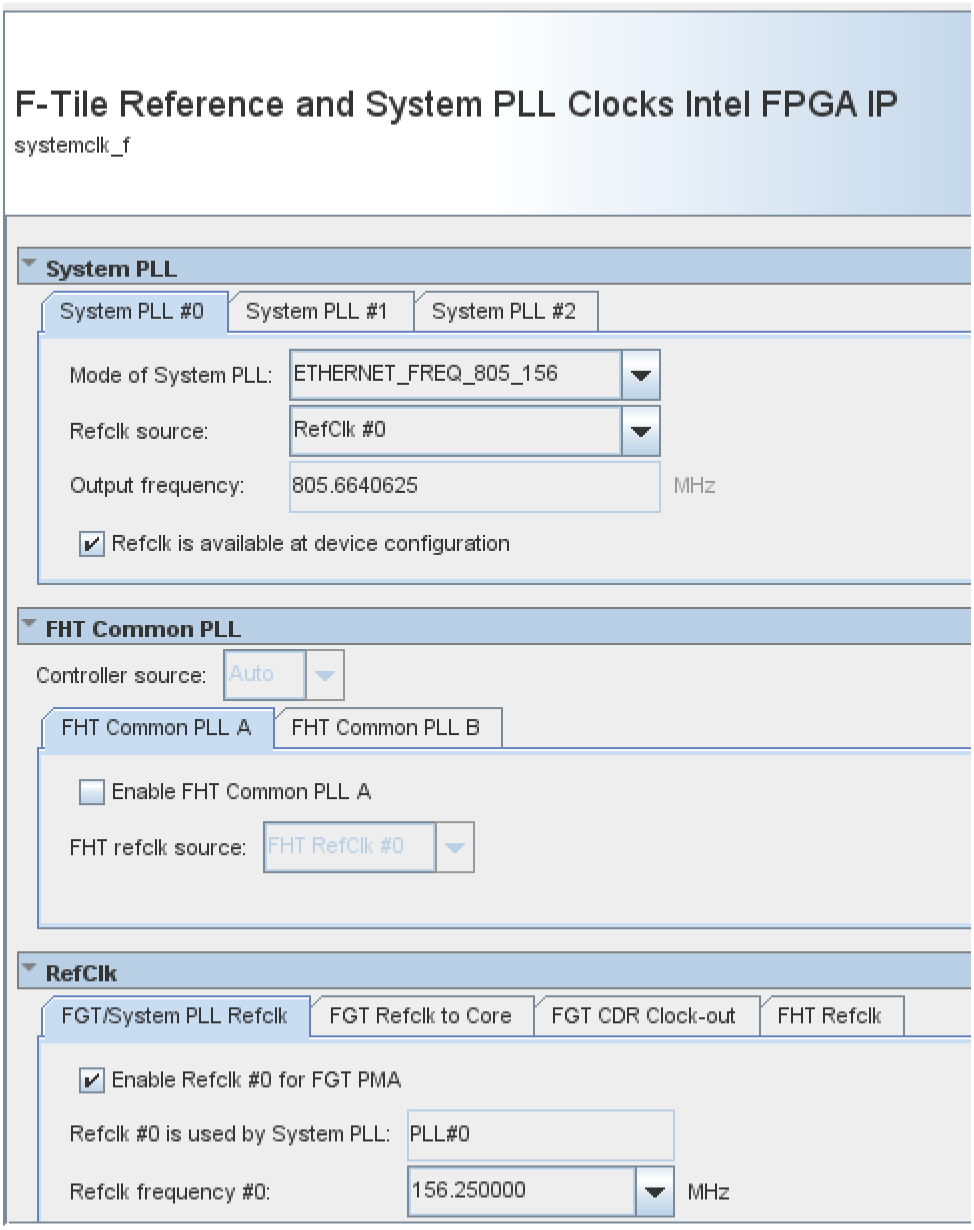


Figure 6-6 The Sytem PLL setting

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	0

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 QSPF28 loopback fixtures.
4. Connect your FPGA board to your PC with a micro 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 Nios II Terminal, as shown in **Figure 6-7.** 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. The loopback test will terminate as shown in **Figure 6-8**.

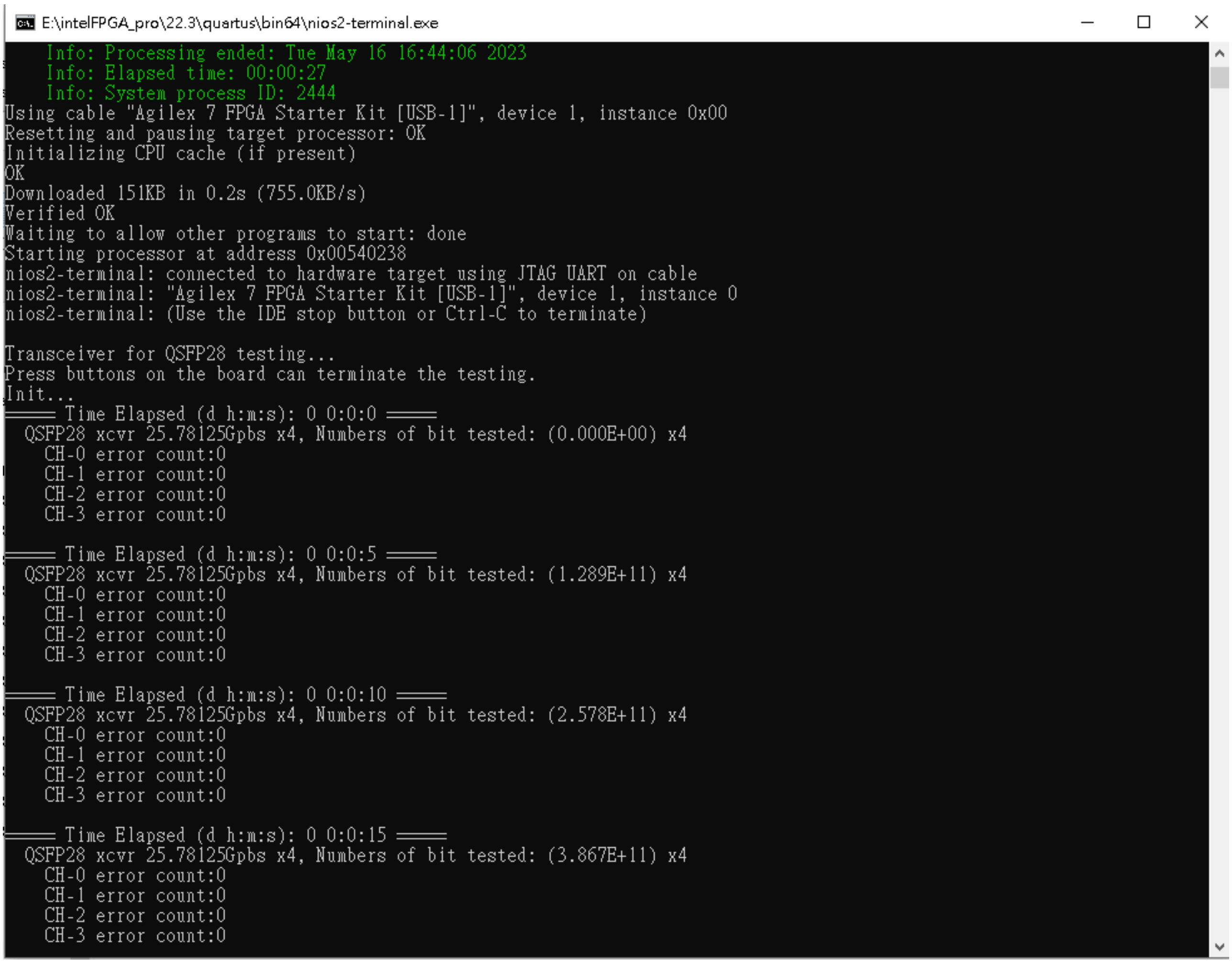


Figure 6-7 QSFP28 Transceiver Loopback Test in Progress

```
F:\intelFPGA_pro\22.3\quartus\bin64\nios2-terminal.exe
Info: Command: quartus_pgm -m jtag -c 1 -o p;..\A7SK.sof
Info (213045): Using programming cable "Agilex 7 FPGA Starter Kit [USB-1]"
Info (213011): Using programming file ../A7SK.sof with checksum 0x75F98086 for d
vr2@1
Info (209060): Started Programmer operation at Wed May 17 10:43:45 2023
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 Wed May 17 10:43:49 2023
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 3353 megabytes
    Info: Processing ended: Wed May 17 10:43:49 2023
    Info: Elapsed time: 00:00:25
    Info: System process ID: 42464
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 0x00440238
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

user abort test!
Transceiver Testing is terminated.
```

Figure 6-8 QSFP28 Transceiver Loopback is terminated

Chapter 7

Additional Information

7.1 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
Agilex 7 FPGA Starter Kit Web: A7SK.terasic.com

■ Revision History

Date	Version	Changes
2023.05	First publication	
2023.07	V1.1	Update section 2.4 and 2.5