

Titan S10 EVK Demonstration Manual



User Manual

FPGA

Contents

Chapter 1	Overview.....	5
Chapter 2	Examples For FPGA	6
2.1	Board Information Query by NIOS V	6
2.2	Board Information IP	9
2.3	DDR4 SDRAM RTL Test	14
2.4	DDR4 SDRAM Test by Nios V.....	17
2.5	USB FX3 Loopback.....	20
Chapter 3	Examples for HPS SoC.....	25
3.1	Network Socket	25
3.2	Build C/C++ Project.....	32
Chapter 4	Examples for using both HPS SoC and FGPA	34
4.1	Required Background	34
4.2	System Requirements.....	35
4.3	AXI bridges in Altera SoC FPGA.....	35
4.4	GHRD Project	37
4.5	Quartus Compile and Programming.....	40
4.6	Develop the C Code.....	40



Chapter 5	<i>PCI Express Reference Design for Windows</i>	45
5.1	PCI Express System Infrastructure	45
5.2	PC PCI Express Software SDK	46
5.3	PCI Express Software Stack	47
5.4	PCI Express Library API	53
5.5	PCIe Reference Design - DDR4	58
Chapter 6	<i>PCI Express Reference Design for Linux</i>	71
6.1	PCI Express System Infrastructure	71
6.2	PC PCI Express Software SDK	72
6.3	Set the Boot Parameters	73
6.4	PCI Express Software Stack	74
6.5	PCI Express Library API	76
6.6	PCIe Reference Design - DDR4	76
Chapter 7	<i>MCIO DDR4</i>	87
7.1	Hardware Setting	87
7.2	PCIe Reference Design – MCIO DDR4	88
Chapter 8	<i>Transceiver Verification</i>	90
8.1	Transceiver Test Code	90
Chapter 9	<i>Additional Information</i>	96

9.1	Getting Help	96
-----	--------------------	----

Chapter 1

Overview

This Manual will introduce the various application demonstrations on **Titan S10 Evaluation Kit**. These demonstrations cover most of the interfaces on the board. Let users familiarize using these interfaces of the board. Demonstrations according to FPGA fabrics and HPS are divided into two categories:

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

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

- [Intel Quartus® Prime Pro Edition Software Version 24.3](#) or later.

Note: To run the demo bath file with the Nios V 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 **Titan S10 Evaluation Kit**. These reference designs cover the features of peripherals connected to the FPGA, such as DDR4, temperature monitor, PLL clock setting and Power monitor. All the associated files can be found in the directory `\Demonstrations\FPGA` of Titan S10 EVK Resource_Package.

2.1 Board Information Query by NIOS V

This demonstration shows how to use the NIOS V processor to query board information from System MAX10 on the SOM and the power monitor IC LTC2945 on the Terasic Titans S10 PCIe carrier board. The board information provided by MAX10 includes power consumption, temperature, and fan speed. LTC2945 is used to monitor the 12V input power status of the Carrier board.

■ System Block Diagram

Figure 2-1 shows the system block diagram of this demonstration. A SPI Master module is used to retrieve board information from MAX10 via SPI interface. NIOS V retrieve this information from the module via GPIO controllers. The 12V Input Power Monitor module is used to query the input power status on the carrier board. NIOS V retrieve the input power status from the module via a GPIO controller.

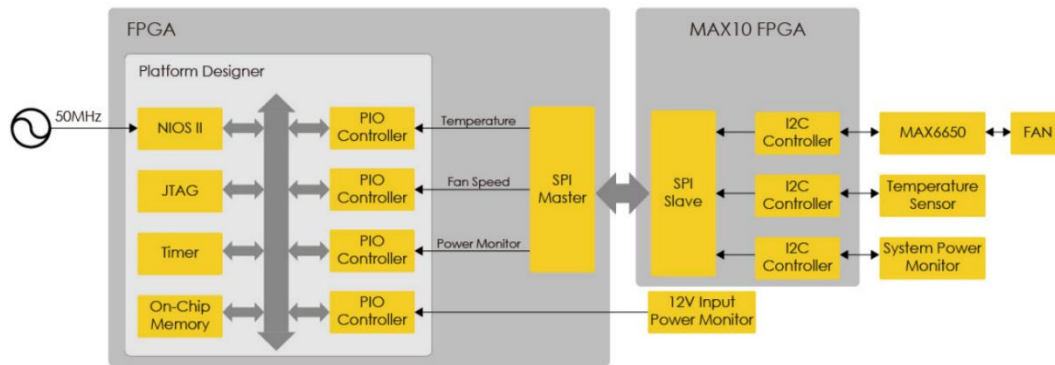


Figure 2-1 Block Diagram of the NIOS Demonstration

The demo provides a sample menu in NIOS terminal, as shown in **Figure 2-2** to provide an interactive interface. With the menu, users can perform the board info information query. Note, pressing '**ENTER**' should be followed by the choice number **0**.

```

C:\WINDOWS\system32\cmd.exe
[GDB server output] Info : [0] Memory access -> Abstract access memory
[GDB server output] Info : [0] CSR & FP Register access -> Abstract commands
[GDB server output]
[GDB server output] Waiting for debugger connection on port 59481.
INFO: Found gdb port 59481
INFO: Starting gdb. Running "riscv32-unknown-elf-gdb -batch -ex set non-stop on
v32 -ex set remotetimeout 60 -ex target extended-remote localhost:59481 -ex moni
load Board_Info.elf -ex set $mstatus &= ~(0x00000088) -ex continue&".
The target architecture is set to "riscv:rv32".
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.

Program received signal SIGINT, Interrupt.
0x00000004 in ?? ()
OK. Software reset asserted
Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x2f4 lma 0x20
Loading section .text, size 0x22da8 lma 0x314
Loading section .rodata, size 0x1848 lma 0x230c0
Loading section .rdata, size 0x1980 lma 0x26288
Start address 0x0000094c, load size 156292
Transfer rate: 433 KB/sec, 22327 bytes/write.
[Inferior 1 (Remote target) detached]
===== Titan S10 Demo Program =====
[0] Display Board Info
Input your choice:

```

Figure 2-2 Menu of Demo Program

In board info test, the program will display temperatures, fan, and power status. Temperatures information includes FPGA temperature, board temperature, and FPGA core power IC temperature. Fan information will display fan rotation speed RPM (rotation per minutes). The power information includes VCC12, FPGA core power, VCC1P8, VCC3P8, VCC09P, VCC1P03

and VCC1P2_IO.

A power monitor IC (LTC2945) embedded on the carrier board monitors real-time current and power. There is a sense resistor R46 (0.003 Ω) for LTC2945 in the circuit. When power is applied to the carrier board, there will be a voltage drop (named Δ SENSE Voltage) on R46. Based on the sense resistors, the power monitor program can calculate the associated voltage, current and power consumption.

■ Demonstration File Location

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

■ Install Ashling RiscFree IDE

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

■ Demonstration Setup and Instructions

1. Make sure Quartus Pro 24.3 or later is installed on the Host PC.
2. Power on the Carrier board.
3. Use the type-C USB Cable to connect your PC and the carrier board. Install USB Blaster III driver if necessary. The driver is available on Quartus Pro 25.1 or later.
4. Execute the demo batch file “test.bat” under the batch file folder:
Board_Info_NiosV\demo_batch.
5. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in command line window.
6. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in NIOS terminal.
7. For temperature monitor, power monitor and fan test, please input key ‘0’ and press ‘Enter’ in the NIOS terminal, as shown in **Figure 2-3**.


```

===== Titan S10 Demo Program =====
[0] Display Board Info
Input your chioce:
0
==== Temperature ====
      FPGA: 35*C
      Board: 38*C
      Power: 0*C

==== Fan ====
      Fan Speed (RPM): 4440

==== Power Monitor ====
VCC12 (Carrier):
      Voltage      = 12.025 V
      Current      = 1.016 A
      Consumption   = 12.217 W
FPGA Core (SOM):
      Voltage      = 0.880 V
      Current      = 0.100 A
      Consumption   = 0.088 W
VCC1P8 (SOM):
      Voltage      = 1.796 V
      Current      = 3.062 A
      Consumption   = 5.499 W
VCC3P3 (SOM):
      Voltage      = 3.300 V
      Current      = 0.700 A
      Consumption   = 2.310 W
VCC0P9 (SOM):
      Voltage      = 0.899 V
      Current      = 0.700 A
      Consumption   = 0.629 W
VCC1P03 (SOM):
      Voltage      = 1.032 V
      Current      = 0.750 A
      Consumption   = 0.774 W
VCC1P2_IO (SOM):
      Voltage      = 1.200 V
      Current      = 0.625 A
      Consumption   = 0.750 W
Display Board Info Test:PASS

```

Figure 2-3 Board Info Demo

2.2 Board Information IP

This section will introduce an IP which can be placed in the Stratix-10 FPGA and allows users to obtain board status information such as power, temperature status and fan speed on the Titan S10 Evaluation Kit.

The kit provides several sensors to monitor the status of the board, such as FPGA temperature, board power monitor, and fan speed status. These interfaces are connected to the system MAX FPGA on the board. The logic in the system MAX FPGA will automatically read the status values

of these sensors and store them in the internal register. As shown in **Figure 2-4**, there is an SPI slave IP in the system MAX FPGA will read the value of the board status from these registers and it can be output to SPI master logic via SPI interface. Another function is that the FPGA reads voltage and current information from the LTC2945 on the Titan S10 Evaluation Kit via the I2C interface, primarily to monitor the voltage and current of the 12V power input.

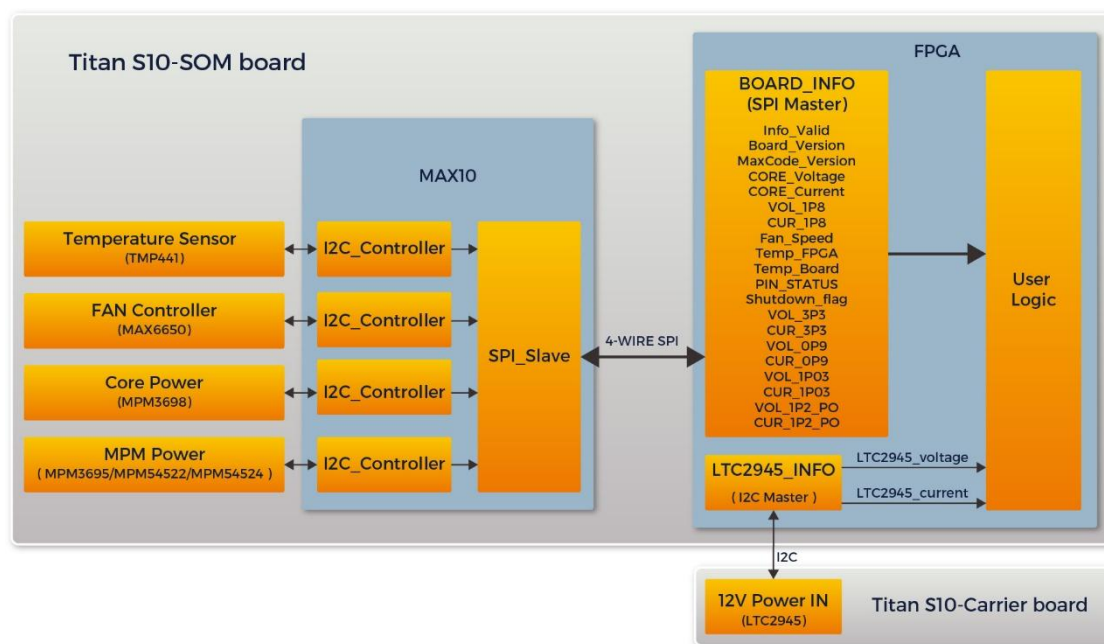


Figure 2-4 Block diagram of the Board Information demonstration

User can placing a board information IP (**BOARD_INFO.v** ; spi master) provided by Terasic in the FPGA, the SOM board status can be obtained via SPI interface from the system MAX FPGA and output to user logic. Another can placing a board information IP (**LTC2945_INFO.v** ; i2c master) provided by Terasic in the FPGA, the Carrier board Power-IN Voltage/Current can be obtained via I2C interface from output to user logic

The board information IP can be obtained from the following path in the Resource_Package:
Demonstration/FPGA/Board_info_RTL/board_information_ip/BOARD_INFO.v
Demonstration/FPGA/Board_info_RTL/LTC2945_ip/LTC2945_INFO.v

Figure 2-5 shows the input and output pins of the board information IPs. Detailed pin descriptions and functions can be obtained from **Table 2-1** Board information IPs input and output ports. The user only needs to provide the IPs 50Mhz clock and the reset control signal. The IPs will automatically communicate with the system MAX(or LTC2945) FPGA to get the

board status value via the SPI/I2C interface. When the logic level of the Info_Valid signal is from low to high, it means that the board status has been updated and can be used.

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

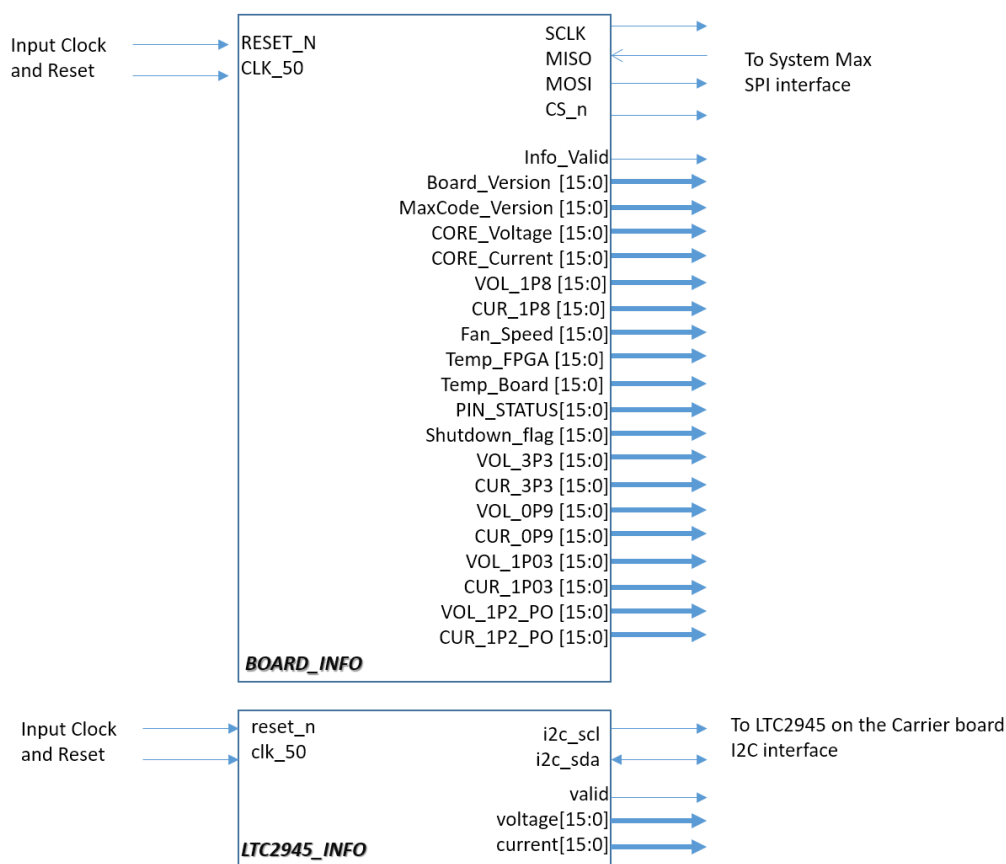


Figure 2-5 Pin out of the board information IP

Table 2-1 Board information IP input and output ports

Port Name	Direction	Width(Bit)	Description
CLK_50	Input	1	Clock input for IP, please input 50Mhz clock.
RESET_N	Input	1	Reset signal for IP, reset all logic.
MOSI	Output	1	Master data output. Please connect this signal to the INFO_SPI_MOSI pin.
MISO	Input	1	Master data input. Please connect this signal to the INFO_SPI_MISO pin.
CS_n	Output	1	Slave Select, Master output. Please connect this signal to the INFO_SPI_CS_n pin.

SCLK	Output	1	Serial Clock, SPI master output to slave. Please connect this signal to the INFO_SPI_SCLK pin.
Info_Valid	Output	1	Information valid, logic high indicates board status updated ready.
Board_Version	Output	16	This information indicates the version of the Titan S10-SOM/Carrier 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.
CORE_Voltage	Output	16	Voltage of the VCC_CORE power channel , Unit is mV
CORE_Current	Output	16	Current of the VCC_CORE power channel , Unit is 10mA
VOL_1P8	Output	16	Voltage of the VCC1P8 power channel , Unit is mV
CUR_1P8	Output	16	Current of the VCC1P8 power channel , Unit is mA
Fan_Speed	Output	16	First fan speed of the board. The unit of the output value is RPM.
Temp_Board	Output	16	First ambient temperature of the development board. The unit of the output value is Celsius.
Temp_FPGA	Output	16	Core FPGA temperature of the development board. The unit of the output value is Celsius.
Shutdown_flag	Output	16	BIT4~15 : Reserved to 0. BIT3: 1 ,when CORE IC Temperature $\geq 120^{\circ}\text{C}$ BIT2: 1 ,when CORE_Current $\geq 65\text{A}$ BIT1: 1 ,when FPGA Temperature $\geq 95^{\circ}\text{C}$ BIT0: 1 ,when Board Temperature $\geq 95^{\circ}\text{C}$
PIN_STATUS	Output	16	BIT8~15 : Reserved to 0. BIT7: FAN_ALERT_n , When the fan speed is abnormal, this bit is 0. BIT6: Reserved to 1.

			BIT5: When shutdown occurs, this bit is 0. BIT4: Reserved to 1. BIT3: Reserved to 0. BIT2: FPGA_CONF_DONE ,FPGA Configure success, this bit is 1. BIT1: Reserved to 1. BIT0: Reserved to 1.
VOL_3P3	Output	16	Voltage of the VCC3P3 power channel , Unit is mV
CUR_3P3	Output	16	Current of the VCC3P3 power channel , Unit is mA
VOL_0P9	Output	16	Voltage of the VCC0P9 power channel , Unit is mV
CUR_0P9	Output	16	Current of the VCC0P9 power channel , Unit is mA
VOL_1P03	Output	16	Voltage of the VCC1P03 power channel , Unit is mV
CUR_1P03	Output	16	Current of the VCC1P03 power channel , Unit is mA
VOL_1P2_PO	Output	16	Voltage of the VCC1P2_PO power channel , Unit is mV
CUR_1P2_PO	Output	16	current of the VCC1P2_PO power channel , Unit is mA
reset_n	Input	1	Reset signal for IP, reset all logic.
clk_50	Input	1	Clock input for IP, please input 50Mhz clock.
i2c_scl	Output	1	Master I2C Clock output. Please connect this signal to the PM_I2C_SCL pin
i2c_sda	IN/OUT	1	Master I2C data. Please connect this signal to the PM_I2C_SDA pin
valid	Output	1	Information valid, logic high indicates board status updated ready.
voltage	Output	16	Voltage of the Carrier-board 12V power channel , Unit is mV
current	Output	16	Current of the Carrier-board 12V power channel , Unit is mA

⊕ BOARD_INFO_i MaxCode_Version[15..0]	0004h
⊕ BOARD_INFO_i CORE_Voltage[15..0]	900
⊕ BOARD_INFO_i CORE_Current[15..0]	250
⊕ BOARD_INFO_i VOL_1P8[15..0]	1788
⊕ BOARD_INFO_i CUR_1P8[15..0]	2937
⊕ BOARD_INFO_i Fan_Speed[15..0]	4440
⊕ BOARD_INFO_i Temp_FPGA[15..0]	37
⊕ BOARD_INFO_i Temp_Board[15..0]	35
⊕ BOARD_INFO_i PIN_STATUS[15..0]	00F7h
⊕ BOARD_INFO_i Shutdown_flag[15..0]	0000h
⊕ BOARD_INFO_i VOL_3P3[15..0]	3300
⊕ BOARD_INFO_i CUR_3P3[15..0]	700
⊕ BOARD_INFO_i VOL_0P9[15..0]	899
⊕ BOARD_INFO_i CUR_0P9[15..0]	550
⊕ BOARD_INFO_i VOL_1P03[15..0]	1032
⊕ BOARD_INFO_i CUR_1P03[15..0]	750
⊕ BOARD_INFO_i VOL_1P2_PO[15..0]	1200
⊕ BOARD_INFO_i CUR_1P2_PO[15..0]	500
u_LTC2945_INFO valid	
⊕ u_LTC2945_INFO voltage[15..0]	12025
⊕ u_LTC2945_INFO current[15..0]	991

Figure 2-6 Waveform of the board status output

2.3 DDR4 SDRAM RTL Test

This demonstration performs a memory test function on the DDR4 memory (DDR4A and DDR4B) on the Titan S10 SOM. The memory size of each DDR4 bank used in this test is 4GB.

■ System Block Diagram

Figure 2-7 shows the function block diagram of this demonstration. There are two DDR4 SDRAM controllers. All of the controllers (DDR4A and DDR4B) use 33.333 MHz as a reference clock. It generates one 1200MHz clock as memory clock from the FPGA to the memory and the controller itself runs at quarter-rate in the FPGA i.e. 33.333 MHz.

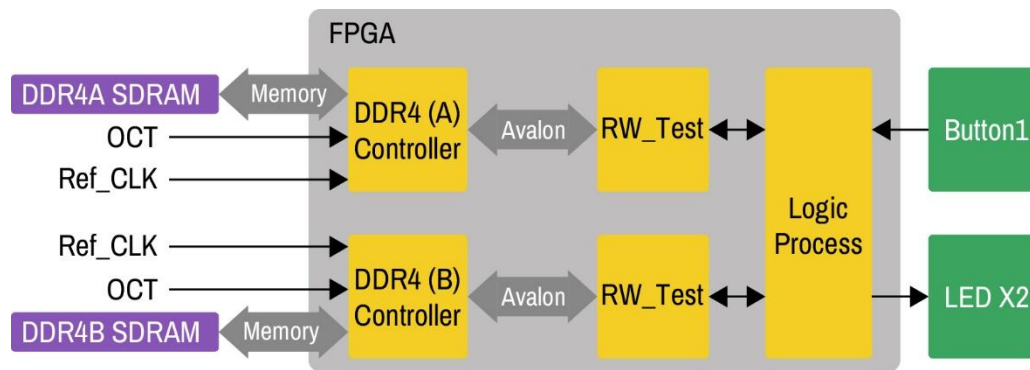


Figure 2-7 Block Diagram of the SDRAM test in Verilog

■ Stratix 10 External Memory Interfaces

To use Stratix 10 External Memory Interfaces controller for DDR4 SDRAM, please perform the two major steps below:

1. Create correct pin assignments for the DDR4 **SDRAM**.
2. Setup correct parameters in the dialog of the Stratix 10 External Memory Interfaces.

■ Design Tools

- Quartus Prime 24.3 Pro Edition or later

■ Demonstration Source Code

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

The demo batch file includes following files:

- ◆ Batch File: test.bat
- ◆ FPGA Configuration File: golden_top.sof

■ Demonstration Setup

1. Make sure Quartus Prime Pro Edition is installed on the Host PC.
2. Connect the Titan S10 Evaluation Kit to the Host PC via the USB cable. Install the USB-Blaster III driver if necessary.
3. Power on the Titan S10 Evaluation Kit.
4. Execute the demo batch file "test.bat" under the batch file folder

\\RTL_DDR4_Test\\demo_batch.

5. Press **BUTTON1** (see **Figure 2-8**) to start DDR4 write & loopback verify process. It will take about 2~3 second to perform the test. While testing, the LED will blink. When LED stop blinking it means the test process is done. In this case, if the LED light, it means the test result is passed. If the LED is no light, it means the test result is failed. The **LED0** represents the test result for the DDR4A, the **LED1** represents the test result for the DDR4B.



Figure 2-8 Location of the KEY and LED on the kit

6. Press **BUTTON1** again to regenerate the test control signals for a repeat test.

Table 2-2 Status of LED Indicators

Name	Description
BUTTON1	Start to test
LED1	ON if the DDR4A test is PASS after releasing BUTTON1
LED2	ON if the DDR4B test is PASS after releasing BUTTON1

2.4 DDR4 SDRAM Test by Nios V

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 on-board DDR4 SDRAM on the FPGA board, and how the Nios V 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 2-9 shows the system block diagram of this demonstration. In the Platform Designer (formerly Qsys), one 100 MHz OSC and two clock generators (Si5332 and Si5341) are used. The OSC and clock generator will provide a 33.333 Mhz clock to the on-board DDR4 SDRAM(DDR4A) as the reference clock. There is a DDR4 Controller which is used in the demonstrations. The controller is responsible for on-board DDR4 SDRAM (DDR4A). The DDR4 controllers are configured as 4GB DDR4 controller. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-3**. The Nios V processor is used to perform the memory test. The Nios V 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.

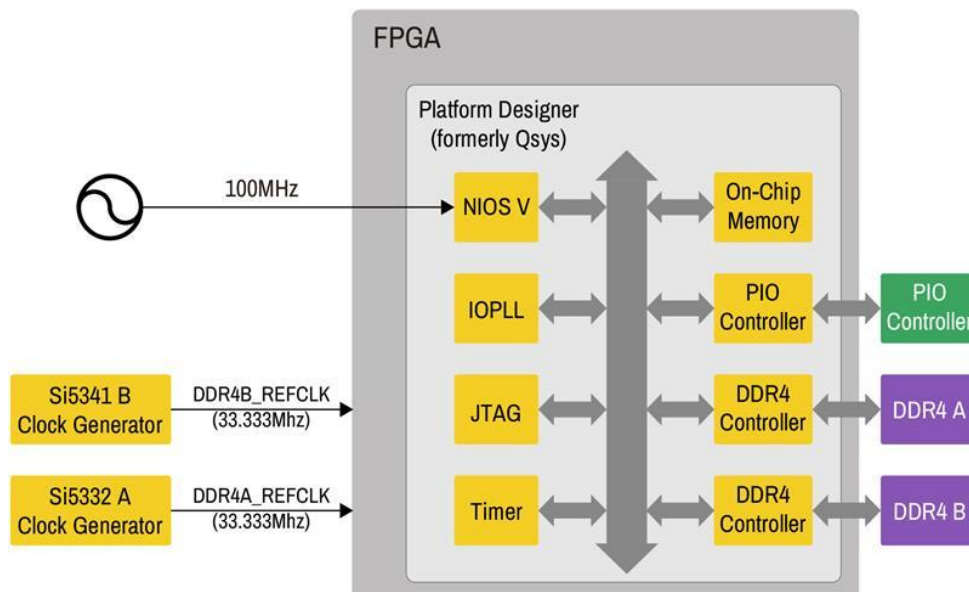


Figure 2-9 Block diagram of the DDR4 Basic Demonstration

The system flow is controlled by a Nios V program. First, the Nios V program writes test patterns into the whole 4GB of SDRAM. Then, it calls Nios V 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 V 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 V terminal when writing/reading data to/from the SDRAM. When verification process is completed, the result is displayed in the Nios V terminal.

Table 2-3 DDR4 clock frequency for each speed grade of FPGA

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
1SX110HN3F43I2V	1200 (DDR4 2400)

■ Design Tools

- Quartus Prime 24.3 Pro Edition

■ Demonstration Source Code

- Quartus Project directory: `DDR4_Test_NiosV`
- Nios V Eclipse: `DDR4_Test_NiosV \software`

■ Demonstration Batch File

Demo Batch File Folder: `DDR4_Test_NiosV\demo_batch`

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat
- FPGA Configure File: golden_top.sof
- Nios V Program: MEM_TEST.elf

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Make sure Quartus Prime and Nios V are installed on your PC.
2. Use a USB Cable to connect the PC and the FPGA board and install USB Blaster III driver if necessary.
3. Power on the FPGA board.
4. Execute the demo batch file “test.bat” under the folder “DDR4_Test_NiosV\demo_batch”.
5. After the Nios V program is downloaded and executed successfully, a prompt message will be displayed in the command-line terminal.
6. For DDR4 Qucik test, please input key ‘1’ and press ‘Enter’ in the command-line terminal as shown in **Figure 2-10**. Press Button0 of the FPGA board to start SDRAM verify process. The program will display progressing and result information.
7. For DDR4 Full test, please input key ‘0’ and press ‘Enter’ in the command-line terminal. The program will display progressing and result information. Press Button0 of the FPGA board to start SDRAM verify process. Note, this test may take about few minutes to run.

```
C:\WINDOWS\system32\cmd.exe
Loading section .text, size 0x252a0 lma 0x40000314
Loading section .rodata, size 0x1910 lma 0x400255c0
Loading section .rwddata, size 0x1a00 lma 0x400288d0
Start address 0x40002630, load size 166084
Transfer rate: 502 KB/sec, 23726 bytes/write.
[Inferior 1 (Remote target) detached]
===== Stratix 10 NIOS DDR4x4 Program =====
[0] DDR4x4 Test
[1] DDR4x4 Quick Test
Input your choice:
1
1
===== DDR4x2 Test! Size=A: 4GB, B: 4GB =====

Press any BUTTON on the board to start test [BUTTON-0 for continued test]
=====> DDR4x2 Testing, Iteration: 1
DDR4x2 Reset durations, 0.977 seconds
DDR4x2 Calibration Duration:0.475 seconds,
== DDR4-A Testing...
DDR4 address bank: 0GB ~ 1GB: PASS
DDR4 address bank: 1GB ~ 2GB: PASS
DDR4 address bank: 2GB ~ 3GB: PASS
DDR4 address bank: 3GB ~ 4GB: PASS
DDR4A test:Pass, 171 seconds
== DDR4-B Testing...
DDR4 address bank: 0GB ~ 1GB: PASS
DDR4 address bank: 1GB ~ 2GB: PASS
DDR4 address bank: 2GB ~ 3GB: PASS
```

Figure 2-10 Quick Test for the DDR4 Basic Demonstration

2.5 USB FX3 Loopback

This demonstration illustrates how the FX3 is working with the FPGA for USB3.0/USB2.0 data bulk in/out (data loop transmission). There is a USB 3.0 A MICRO B cable connector onto Titan-S10-SOM Carrier Platform, a USB 3.0 A MICRO B cable is reversible for plugging in the USB3.0 MICRO B connector. This demonstration also implements the auto-switching mechanism for a USB 3.0 A MICRO B cable plugging in on either side.

■ Function Block Diagram

Figure 2-11 shows the function block diagram of the USB FX3 demonstration. This design comprises two parts, FX3 switch controlling and FX3 data transferring. For the data transmission of FX3 module, FIFO and controller (implemented in the FPGA) combine FX3 module to perform the data bulk in/out loop. (For details, please refer to CYPRESS AN65974 Designing with the EZ-USB® FX3™ Slave FIFO Interface Chapter 11). All modules functions are described below:

LOOPBACK: This module is designed as FX3 Slave FIFO Interface, the module combines the CYPRESS application(bulkloop.exe) to implement data bulk in/out loop demo.

BUTTON1: It is used to reset FX3.

SW0: The switch is connected to the FPGA, which then wires to the USBFX3_USB_MODE

signal. This connects to the MAX10, which is used to configure the FX3's PMODE[2:0] pins. When SW0 = 1, PMODE[2:0] = {1'bz, 1'b1, 1'b1} (F11), and the FX3 enters USB boot mode. When SW0 = 0, PMODE[2:0] = {1'b0, 1'bz, 1'b1} (0F1), and the FX3 enters SPI boot mode

The Titan S10-SOM Platform has a 4Mbits Flash ROM, which can be used to program the FX3 firmware. This ROM is connected to FX3 through SPI interface. PMODE[2:0] is used to set the FX3 in program or boot status. The setting details is described in below steps.

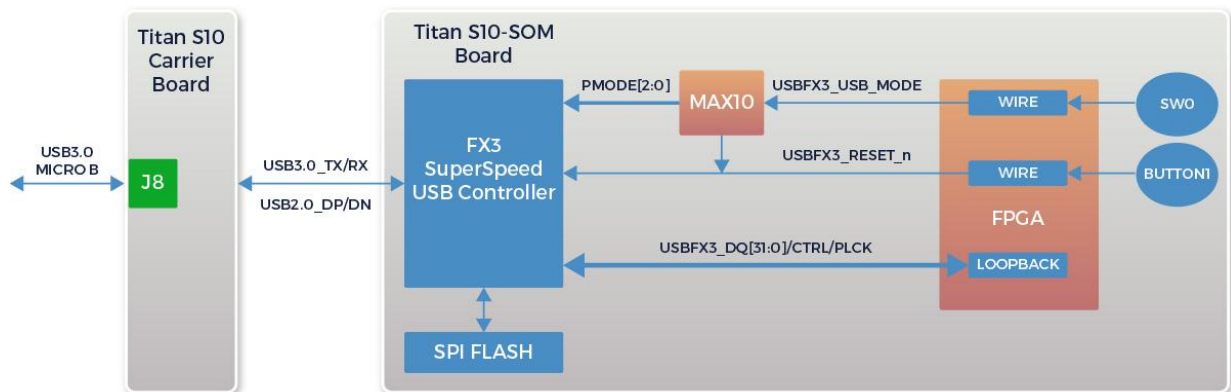


Figure 2-11 Block diagram of the USB FX3 design

■ Design Tools

- Quartus Prime 24.3 Pro Edition

■ Demonstration Source Code

- Quartus Project directory: FX3
- Bitstream used: golden_top.sof

■ Demonstration Batch File

- Demo Batch File Folder: Demonstrations\FPGA\FX3\demo_batch

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Connect the Titan S10-Carrier Platform USB3 Type-C connector (J10) to the host PC with a USB cable and install the USB III driver if necessary.
2. Use a USB 3.0 A MICRO B cable to connect the Titan Carrier Platform (J9) and a PC (with a USB3.0 connector). As shown in **Figure 2-12**.

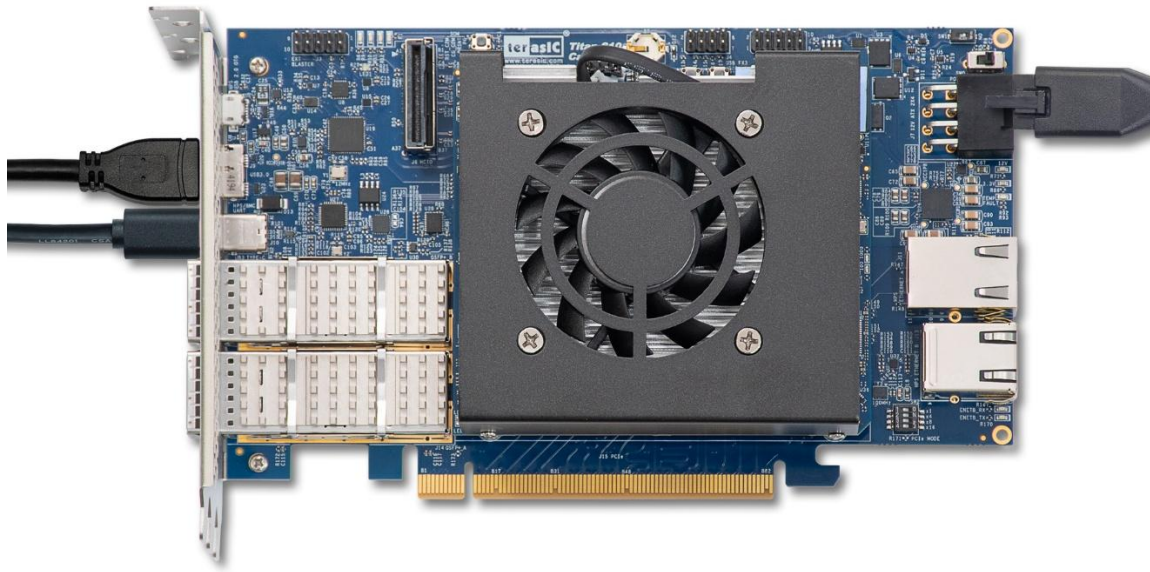


Figure 2-12 USB FX3 Demo Hardware Setting Up

3. Plug the 12V adapter to Titan Carrier Platform DC 12V power connector (J7).
4. Power on the Titan S10-Carrier Platform.
5. Execute the demo batch file "test.bat" from the directory \FPGA\FX3\demo_batch.
6. Install the FX3 driver: the driver for Windows 10 is in the \FPGA\FX3\demo_batch\Driver\win10 folder.
7. Use SW0=0 to set the PMODE[2:0] as "0F1" (F indicates floating).
8. Press BUTTON1 (RESET FX3).
9. Re-plug the USB 3.0 A MICRO B cable one time, Cypress Control Center (FPGA\FX3\demo_batch\Host_app\download_firmware\CyControl.exe) will show Cypress FX3 USB Streamer Example Device and BOS (SuperSpeed Device capability), it indicates the USB3.0 Has completed the setup.
10. Execute the .exe application: \demo_batch\Host_app\loopback\bulkloop.exe, see Figure 2-12 below, press Start, you will see the Bytes Transferred IN/Out value increasing rapidly. For more information of bulkloop.exe, please refer to Cypress EZ-USB FX3 SDK Getting Started with FX3 SDK.

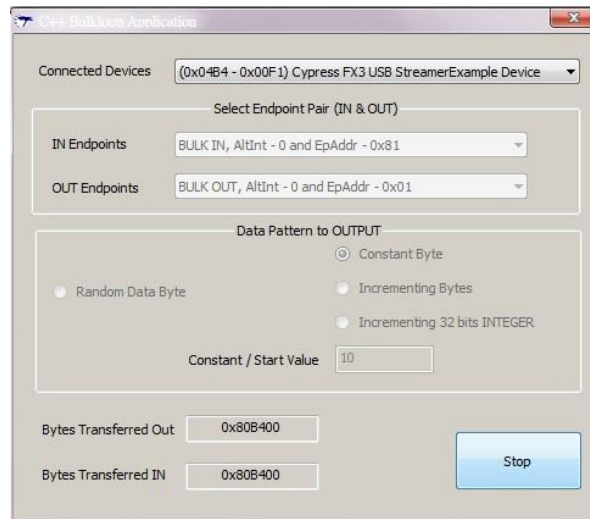


Figure 2-13 FX3 bulk in out bulkloop

Table 2-4 The functional keys of the USBC_FX3 demonstration

Slide Switches	0 – DOWN Position
SW0	SW0=1 , FX3 PMODE[2:0]= { 1'bz, 1'b1, 1'b1 }(F11) , FX3 is boot mode. SW0=0 ,FX3PMODE[2:0]= { 1'b0, 1'bz, 1'b1 }(0F1) FX3 is SPI mode.
BUTTON1	It is used to reset FX3 module.

■ Program the FX3 firmware(Optional)

1. Execute the demo batch file “test.bat” from the directory \FPGA\FX3\demo_batch
2. Execute Cypress Control:
FPGA\FX3\demo_batch\Host_app\ download_firmware\CyControl.exe
3. Use SW0=1 to set the PMODE[2:0] as “F11”(F indicates floating).
4. Press BUTTON1 (RESET FX3), Cypress Control center will show Cypress FX3 USB Bootloader Device.
5. Program software, In Control Center. Click Program FX3 SPI FLASH, as shown in **Figure 2-14**. Select file FPGA\FX3\demo_batch\FX3_Firmware\SF_loopback.img. Wait until It reports “Programming of SPI FLASH Succeeded”.

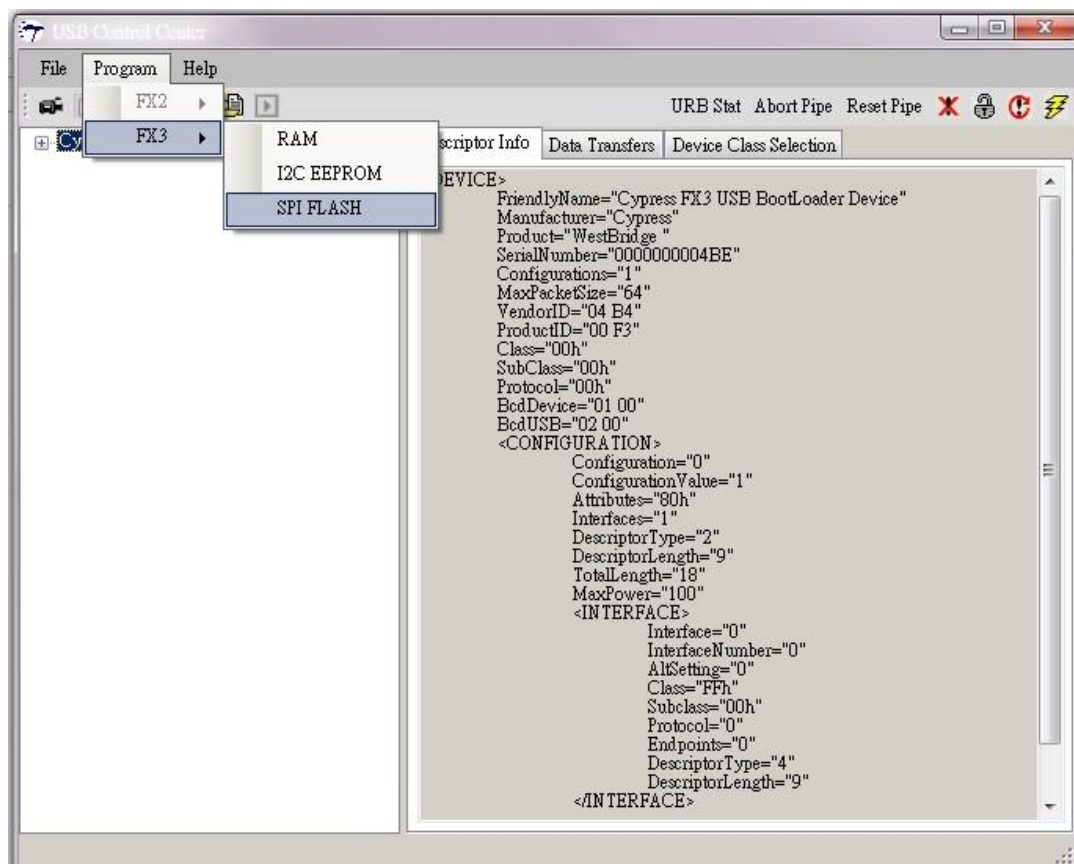


Figure 2-14 Cypress Control Center

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 Network Communication. All the associated files can be found in the directory /Demonstrations/SOC of the Resource Package.

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 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 3-1**. 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.

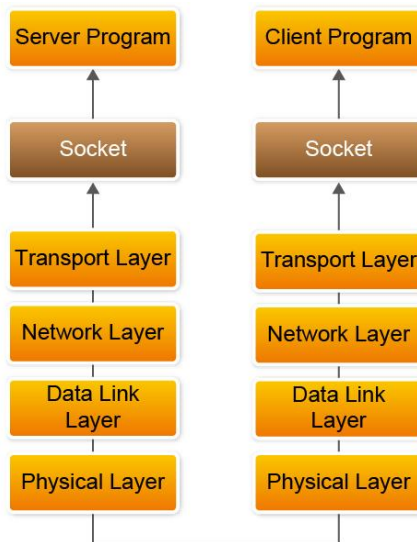


Figure 3-1 Communicate on a network via a socket

■ 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 3-2** shows the communication diagram between the client and server.

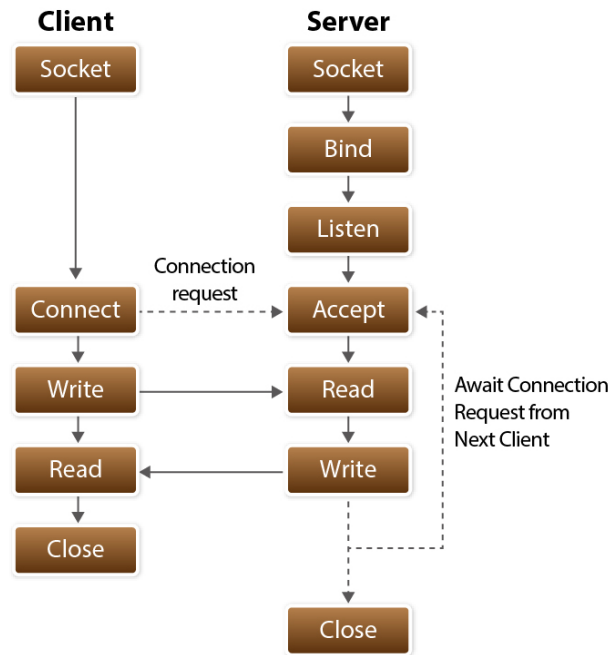


Figure 3-2 Client and Server communication

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 3-3** 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 3-3 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**

3-4 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 remote socket server based on the given Hostname (or IPv4 Address) and port number. Data receiving and sending is implemented by **read** and **write** API, and **close** is used to **close** the socket.

```

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr, "ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr.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 3-4 Socket Client Code

■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-5**. The Demonstration folder contains three platform subfolders: **arm** and **linux**. 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. Each platform subfolder contains **socket_client** and **socket_server** project folders.

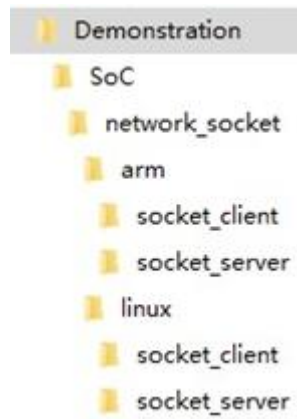


Figure 3-5 Source Code Folder Tree

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 Board to Network via Ethernet port (J11).
2. Connect a USB cable to the Type-C 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 Board Linux BSP micro SD card into the board.
5. Power on the board.
6. Launch the Putty to connect board via the USB-to-UART link.
7. In the Putty, type user name "**root**" 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 3-6**. The port number can be any value between 2000 and 63500.

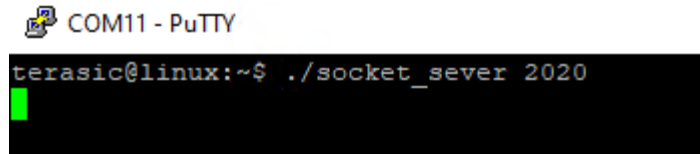


Figure 3-6 Start Socket Server

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 3-7**.

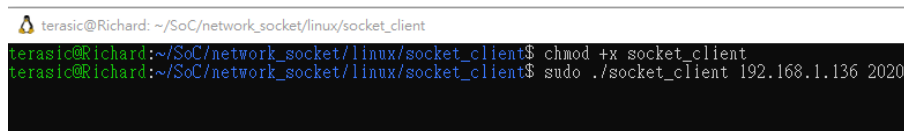


Figure 3-7 Start Client Program

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 3-8**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-9**.

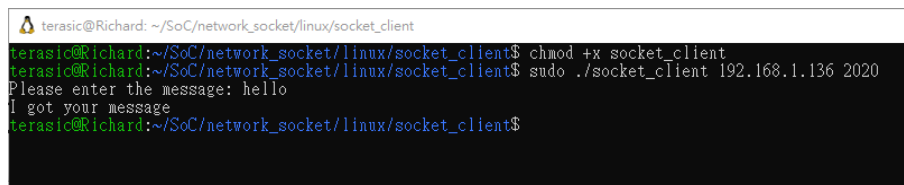
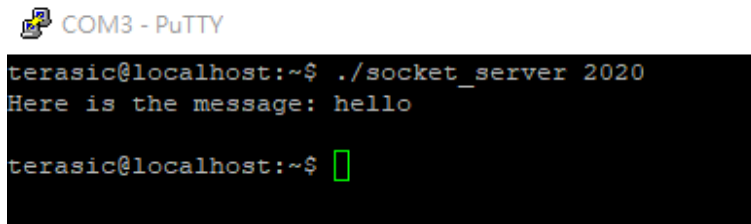


Figure 3-8 Send Message in Client Program



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

Figure 3-9 Server dumps received message

3.2 Build C/C++ Project

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

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 Resource_Package:

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 Demo project into the Linux System and go to the project folder.
6. Type “make” to build project as shown in **Figure 3-10**.


```

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 3-10 Build C/C++ Project

Chapter 4

Examples for using both HPS SoC and FGPA

This Chapter demonstrates how to use the HPS/ARM to communicate with FPGA. We will introduce the GHRD project for Atum A5 development board. And we develop one ARM C Project which demonstrates how HPS/ARM program controls the three of four LEDs connected to FPGA. We will show how HPS controls the FPGA LED through Lightweight HPS-to-FPGA Bridge. The FPGA is configured by HPS through FPGA manager in HPS.

4.1 Required Background

This section pre-assumed the developers have the following background knowledge:

■ **FPGA RTL Design**

- Basic Quartus Prime Pro operation skill
- Basic RTL coding skill
- Basic Platform Designer operation skill
- Knowledge about Memory-Mapped Interface

■ **C Program Design**

- Basic C coding and compiling skill
- Skill to Create a Linux Boot SD-Card with a given HPS Linux image file
- Skill to boot Linux from SD-Card and copy files into HPS Linux file system from host PC

4.2 System Requirements

Before starting this tutorial, please note that the following items are required to complete the demonstration project:

■ Terasic FPGA-SOC board, includes

- Type-A to Type-C USB Cable for UART terminal
- Micros SD-Card, at 4GB minimum
- Micros SD-Card Card Reader

■ A x86 PC

- Linux or Windows with WSL installed
- One USB Port
- Quartus Prime Pro 24.3 or Later Installed
- ARM GNU/Linux Toolchain installed
- Win32 Disk Imager Installed

4.3 AXI bridges in Altera SoC FPGA

In Altera SoC FPGA, the HPS logic and FPGA fabric are connected through the AXI (Advanced eXtensible Interface) bridge. For HPS logic to communicate with FPGA fabric, Altera system integration tool Platform Designer should be used for the system design to add HPS component. From the AXI master port of the HPS component, HPS can access those Platform Designer components whose memory-mapped slave ports are connected to the master port.

The HPS contains the following HPS-FPGA AXI bridges.

- FPGA-to-HPS Bridge
- HPS-to-FPGA Bridge
- Lightweight HPS-to-FPGA Bridge

Figure 4-1 shows a block diagram for the AXI-4 bridges between FPGA fabric and the level 3 (L3) interconnect to the HPS.

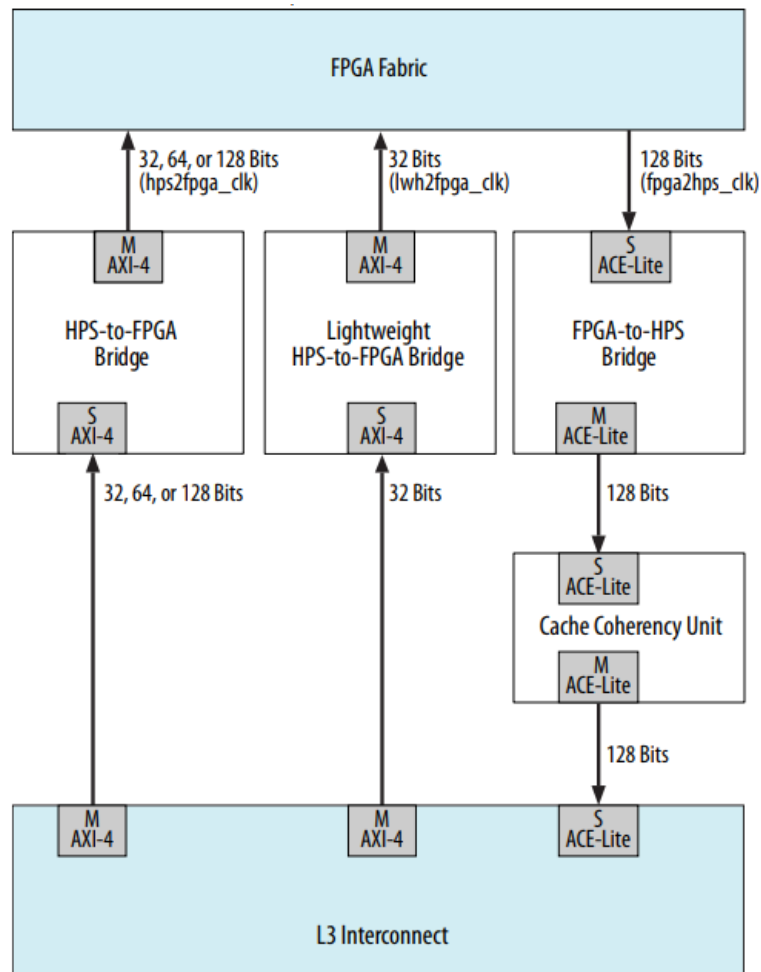


Figure 4-1 AXI-4 Bridge Block Diagram

The HPS-to-FPGA and lightweight HPS-to-FPGA bridges are both mastered by the L3 interconnect, while the FPGA-to-HPS bridge is a master to the CCU. This arrangement allows any master implemented in the FPGA fabric to access most slaves in the HPS.

The HPS-to-FPGA bridge provides a configurable-width, high-performance master interface to the FPGA fabric. The bridge provides most masters in the HPS with access to logic and peripherals implemented in the FPGA. The size of the address space is 4 GB. You can configure the bridge master exposed to the FPGA fabric for 32/64/128-bit data.

The lightweight HPS-to-FPGA bridge provides a lower-performance interface to the FPGA fabric. This interface is useful for accessing the control and status registers of soft peripherals. The bridge provides a 2 MB address space and access to logic, peripherals, and memory implemented in the FPGA fabric. The Cortex-A53 MPCore processor, direct memory access

(DMA) controller, and debug access port (DAP) can use the lightweight HPS-to-FPGA bridge to access the FPGA fabric or NoC registers.

The FPGA-to-HPS bridge provides access to the peripherals in the HPS from the FPGA. This access is available to any master implemented in the FPGA fabric. You can configure the bridge slave, which is exposed to the FPGA fabric, to support the ACE-Lite protocol, with a data width of 128 bits.

This Demo introduces to users how HPS access the FPGA LED via Lightweight HPS-to-FPGA (LWH2F) bridge.

4.4 GHRD Project

The term GHRD is short for Golden Hardware Reference Design. The GHRD project provide by Terasic for the Atum A5 development board is located in the Resource Package folder: \Demonstration\SOC_FPGA\GHRD.

The project consists of the following components:

- HPS Subsystem
- FPGA Peripheral Subsystem, include
 - Two user push-button inputs
 - Two user DIP switch inputs
 - One user I/O for LED outputs
 - 256KB on-chip memory

The memory map of system peripherals in the FPGA portion of the SoC as viewed by the ARM microprocessor unit (MPU) starts at the lightweight HPS-to-FPGA base address 0x00_F900_0000 as shown in **Figure 4-2**. The MPU can access these peripherals through the Address offset setting in the Platform Designer. User can open the GHRD project with Quartus II Software. Then open the qsys_top.html file with an Internet Brower tool as shown in **Figure 4-3**. In the content, you can find the FPGA LED controller periph_led_pio address is 0x0000_1080.

Note, this LED address 0x0000_1080 will be used in following demo C code.

Absolute Address	Region Size	Physical Address	
132 GB		0x20_FFFF_FFFF	
	4 GB	0x20_0000_0000	FPGA
128 GB		0x1F_FFFF_FFFF	
	124 GB	0x01_0000_0000	Memory
4 GB			
	224 KB		Hole
		0x00_FFFC_7FFF	
	32 KB	0x00_FFFC_1000	gicspace
	768 KB		Hole
		0x00_FFEF_FFFF	
	1 MB	0x00_FFE0_0000	OCRAM
		0x00_FFDF_FFFF	
	110 MB	0x00_F900_0000	iospace1
		0x00_F8FF_FFFF	
	16 MB	0x00_F800_0000	DDR Registers
		0x00_F7FF_FFFF	
	16 MB	0x00_F700_0000	CCU Registers
	368 MB		Hole
		0x00_DFFF_FFFF	
	1.5 GB	0x00_8000_0000	FPGA
		0x00_7FFF_FFFF	
	2 GB	0x00_0000_0000	Memory
2 GB			

hps_per (6 MB)	0x00_FF80_0000
dap (8 MB)	0x00_FF00_0000
Hole	0x00_FD00_0000
stm (16 MB)	0x00_FC00_0000
Hole	0x00_FB00_0000
tcu (16 MB)	0x00_FA00_0000
Hole	0x00_F920_0000
lws2f (2 MB)	0x00_F900_0000

Figure 4-2 L3 Address Regions

2025.05.14 14:48:28

All Components

- fpga_m2ocm_pb** altera_avalon_mm_bridge 20.1.0
- mge_led_pio** altera_avalon_pio 19.2.3
- mge_rcfg_pio** altera_avalon_pio 19.2.3
- ocm** altera_avalon_onchip_memory2 19.3.9
- syzid** altera_avalon_syzid_qsys 19.1.7
- hps_mge** subsys_mge 1.0
- hps_mge_alt_mge_phy_1** alt_mge_phy 24.0.0
- hps_mge_hps_to_mge_gmii_adapter_1** hps_to_mge_gmii_adapter 19.2.3
- hps_mge_pb_0** altera_avalon_mm_bridge 20.1.0
- hps_mge_pb_mge_rcfg_0** altera_avalon_mm_bridge 20.1.0
- periph** subsys_periph 1.0
- periph_ILC_interrupt_latency_counter** 19.1.2
- periph_button_pio** altera_avalon_pio 19.2.3
- periph_dipsw_pio** altera_avalon_pio 19.2.3
- periph_led_pio** altera_avalon_pio 19.2.3
- periph_pb_cpu_0** altera_avalon_mm_bridge 20.1.0

All the Avalon Conduit signals of these peripherals are connected to the I/O pins of the SoC FPGA on Atum A5 as shown in the **Figure 4-4**.

```
// Qsys Top module
qsys_top soc_inst (
    .src_prb_rst_sources_source      (src_reset_n),
    .reset_reset_n                  (~system_reset_sync),
    .clk_100_clk                     (system_clk_100_internal),
    .ninit_done                     (ninit_done),
    .led_pio_external_connection_in_port (fpga_led_internal),
    .led_pio_external_connection_out_port (fpga_led_internal),
    .dipsw_pio_external_connection_export (fpga_dipsw_pio),
    .button_pio_external_connection_export (fpga_debounced_buttons),
    .s100_hps_f2h_stm_hw_events_stm_hwevents (stm_hw_events),
```

Figure 4-4 Connection in the top design

4.5 Quartus Compile and Programming

In the Platform Designer, click the menu item “Generate→Generate...” to generate source code for the system and then close the Platform Designer. Now, users can start the compile process by clicking the menu item “Processing→Start Compilation”. When the compilation process is completed successfully, **golden_top_hps.sof** is generated in the GHRD\output_files folder. Users can use this file to configure FPGA by Quartus Programming through the on-board USB-Blaster.

4.6 Develop the C Code

This section introduces how to design an ARM C program to control the led_pio PIO controller. ARM GNU/Linux Toolchain is used to compile the C project. For ARM program to control the led_pio PIO component, led_pio address is required. The Linux built-in driver ‘/dev/mem’ and mmap system-call are used to map the physical base address of led_pio component to a virtual address which can be directly accessed by Linux application software.

■ Map LED_PIO Address

This section will describe how to map the pio_led physical address into a virtual address which is accessible by an application software. **Figure 4-5** shows the C program to derive the virtual address of led_pio base address. First, open system-call is used to open memory device driver “/dev/mem”, and then the mmap system-call is used to map HPS physical address into a virtual address represented by the void pointer variable virtual_base. The demo code maps the physical base address (LWHPS2FPGA_BASE = 0xF9000000) of the peripheral region into a based virtual address **virtual_base**. For any controller in the peripheral region, users can calculate their virtual address by adding their offset relative to the peripheral region to the based virtual address **virtual_base**. Based on the rule, the virtual address **h2p_lw_led_addr** of **periph_led_pio** can be calculated by adding the offset address (0x0000_1080) of **periph_led_pio** to the virtual_base.


```

// Stratix 10 HPS Register Map
// https://www.intel.com/content/www/us/en/programmable/hps/stratix-10/hps.html

//////// HPS2FPGA LW Bridge //////////
// FPGA_bridge_lwsoc2fpga_2M
// https://www.intel.com/content/www/us/en/programmable/hps/stratix-10/hps.html
#define LWHPS2FPGA_BASE ( 0xf9000000 ) //LWHPS2FPGA_memory
#define LWHPS2FPGA_SPAN ( 0x200000 ) // 2MB
#define LWHPS2FPGA_MASK ( LWHPS2FPGA_SPAN - 1 )

#define periph_led_pio 0x1080 // GHRD/qsys_top/qsys_top.html
#define periph_led_mask 0x01

int main() {
    void* virtual_base;
    int fd;
    int loop_count;
    //int led_direction;
    int led_mask;
    void* h2p_lw_led_addr;
    const int blink_dur_sec = 30;

    // map the address space for the LED registers into user space so we can interact with them.
    // we'll actually map in the entire CSR span of the HPS since we want to access various registers within that span

    if ((fd = open("/dev/mem", (O_RDWR | O_SYNC)) == -1) {
        printf("ERROR: could not open \"/dev/mem\"...\n");
        return(1);
    }
    //printf("LWHPS2FPGA_SPAN=0x%x\n", LWHPS2FPGA_SPAN);
    virtual_base = mmap(NULL, LWHPS2FPGA_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd, LWHPS2FPGA_BASE );

    if (virtual_base == MAP_FAILED) {
        printf("ERROR: mmap() failed...\n");
        printf("errno : %s\n", strerror(errno));
        close(fd);
        return(1);
    }

    h2p_lw_led_addr = virtual_base + ((unsigned long)(periph_led_pio) & (unsigned long)(LWHPS2FPGA_MASK ));
}

```

Figure 4-5 LED_PIO memory map code

■ LED Control

C programmers need to understand the Register Map of the PIO core for LED_PIO before they can control it. **Figure 4-6** shows the Register Map for the PIO Core. Each register is 32-bit width. For detail information, please refer to the datasheet of PIO Core. For led control, we just need to write output value to the offset 0 register relative to based address 0x0000_1080. Because the led on the FPGA-SoC board is low active, writing a value 0x00000000 to the offset 0 register will turn on LEDs. There are four green LEDs on Atum A5 and only three (LED0~LED2) of them are connected to this PIO controller. The last LED (LED3) is used to implement FPGA heartbeat. Writing a value 0x00000000 to the offset 0 register will turn on all of nine red LEDs. In C program, writing a value 0x00000000 to the offset 0 register of periph_led_pio can be implemented as:

```
*(uint32_t *) h2p_lw_led_addr= 0x00000000;
```

The state will assign the void pointer to a uint32_t pointer, so C compiler knows write a 32-bit value 0x00000000 to the virtual address h2p_lw_led_addr.

Offset	Register Name		R/W	(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs				
		write access	W	New value to drive on PIO outputs				
1	direction (1)		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask (1)		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture (1) , (2)		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set. Outset value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				
5	outclear		W	Specifies which output bit to clear. Outclear value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				
Note :								
1. This register may not exist, depending on the hardware configuration. If a register is not present, reading the register returns an undefined value, and writing the register has no effect.								
2. If the option Enable bit-clearing for edge capture register is turned off, writing any value to the <code>edgecapture</code> register clears all bits in the register. Otherwise, writing a 1 to a particular bit in the register clears only that bit.								

Figure 4-6 Register Map of PIO Core

■ Main Program

In the main program, the LED is controlled to perform LED light shifting operation as shown in **Figure 4-7**. When finishing 60 times of shift cycle, the program will be terminated.

```
// blink LED. one per second. total blink 30 second
loop_count = 0;
led_mask = periph_led_mask;
printf("blink led %d seconds\r\n", blink_dur_sec);
while (loop_count < blink_dur_sec*2) {
    printf("led %s\r\n", (led_mask & periph_led_mask)? "on": "off");
    // control led
    *(uint32_t*)h2p_lw_led_addr = ~led_mask;

    led_mask = ~led_mask;
    loop_count ++;

    // wait 500ms
    usleep(500 * 1000);
}
```

Figure 4-7 C Program for LED Blink Operation

■ Makefile and compile

Figure 4-8 shows the content of Makefile for this C project. In the Makefile, ARM-linux cross-compile also be specified.

```

TARGET = HPS_FPGA_LED
C_SRC := main.c
CFLAGS := -g -O0 -Werror -Wall

ARCH=arm64
CC := $(CROSS_COMPILE)gcc
NM := $(CROSS_COMPILE)nm

ifeq ($(or $(COMSPEC),$(ComSpec)),)
RM := rm -rf
else
ifeq ($(TERM),cygwin)
RM := rm -rf
else
RM := del
endif
endif

ELF ?= $(TARGET)
#ELF ?= $(basename $(firstword $(C_SRC)))
OBJ := $(patsubst %.c,%.o,$(C_SRC))

.PHONY: all
all: $(ELF)

.PHONY:
clean:
    $(RM) $(ELF) $(OBJ) *.objdump *.map

$(OBJ): %.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

$(ELF): $(OBJ)
    $(CC) $(CFLAGS) $(OBJ) -o $@ $(LDFLAGS)
    $(NM) $@ > $@.map

```

Figure 4-8 Makefile content

To compile the project, please make sure the ARM GNU/Linux tool chain is installed. Please refer section “[3.3 Build C/C++ Project](#)” to install the tool.

To compile the project, type “make” in the command shell as shown in **Figure 4-9**. Then, type “ls” to check the generated ARM execution file “HPS_FPGA_LED”.

```

terasic@Richard:~/HPS_FPGA_LED$ ls
Makefile main.c
terasic@Richard:~/HPS_FPGA_LED$ 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 main.o -o HPS_FPGA_LED
aarch64-none-linux-gnu-nm HPS_FPGA_LED > HPS_FPGA_LED.map
terasic@Richard:~/HPS_FPGA_LED$ ls
HPS_FPGA_LED HPS_FPGA_LED.map Makefile main.c main.o
terasic@Richard:~/HPS_FPGA_LED$

```

Figure 4-9 ARM C Project Compilation

■ Execute the Demo

1. To execute the demo, please boot the Linux from the SD-card in the FPGA-SOC board. Copy the execution file “HPS_FPGA_LED” to the Linux directory, and type “chmod +x HPS_FPGA_LED” to add execution attribute to the execute file. Use Quartus Programmer to configure FPGA with the golden_top_hps.sof generated in previous chapter. The LED1 will flash as the heart beat of the FPGA. Then, type “./HPS_FPGA_LED” (when login with root account) to launch the ARM program. The LED0 on will be expected to perform 30 times of LED blink operation, and then the program is terminated

Chapter 5

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 Altera FPGA IP for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel document [ug20297-683821-844753](https://www.intel.com/content/dam/develop/external/us/en/documents/ug20297-683821-844753.pdf).

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 Altera FPGA IP for PCI Express. The application software on the PC side is developed by Terasic based on Intel's PCIe kernel mode driver.

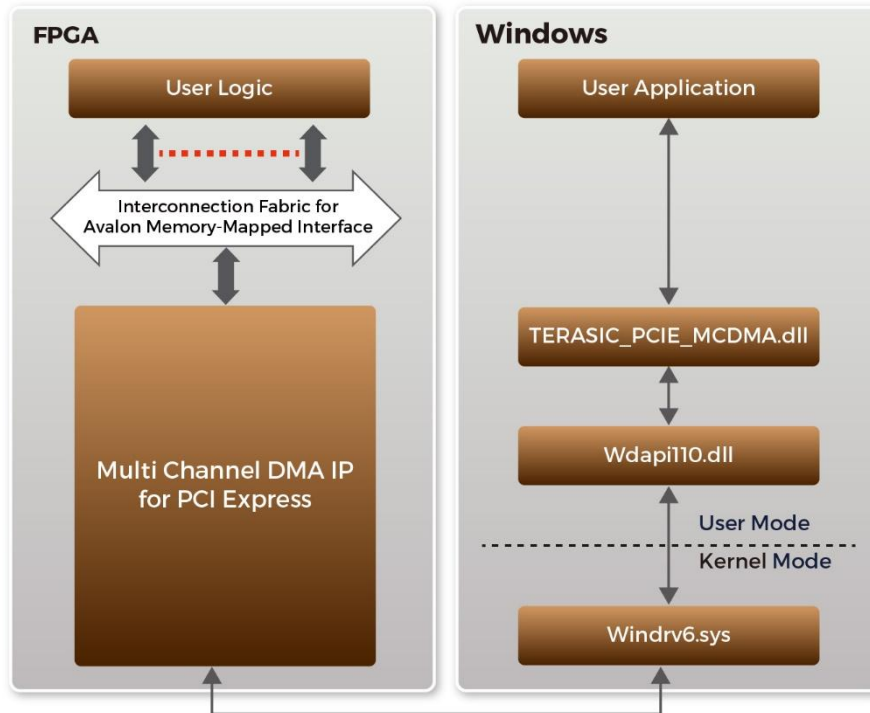


Figure 5-1 Infrastructure of PCI Express System

5.2 PC PCI Express Software SDK

The FPGA Resource_Package 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 "Resource_Package\Demonstration\FPGA\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.

5.3 PCI Express Software Stack

Figure 5-2 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.

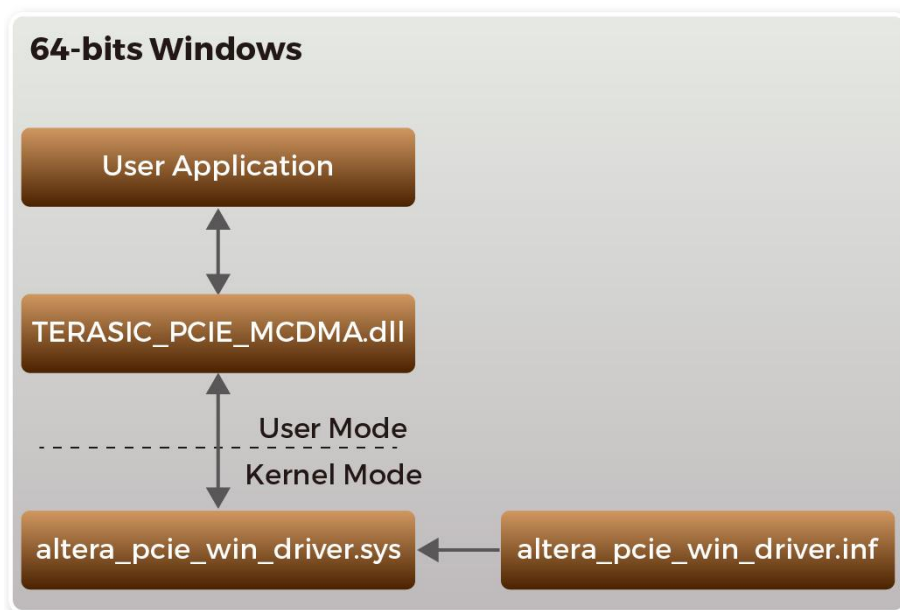


Figure 5-2 PCI Express Software Stack

■ Install PCI Express Driver on Windows

The PCIe driver is located in the folder:

"Resource_Package\Demonstration\FPGA\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 `"Resource_Package\Demonstration\FPGA\PCIe_DDR4\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 two PCI Devices item in the dialog, as shown in **Figure 5-3**. Move the mouse cursor to the PCI Device item and right click it to select the Updated Driver Software... items.

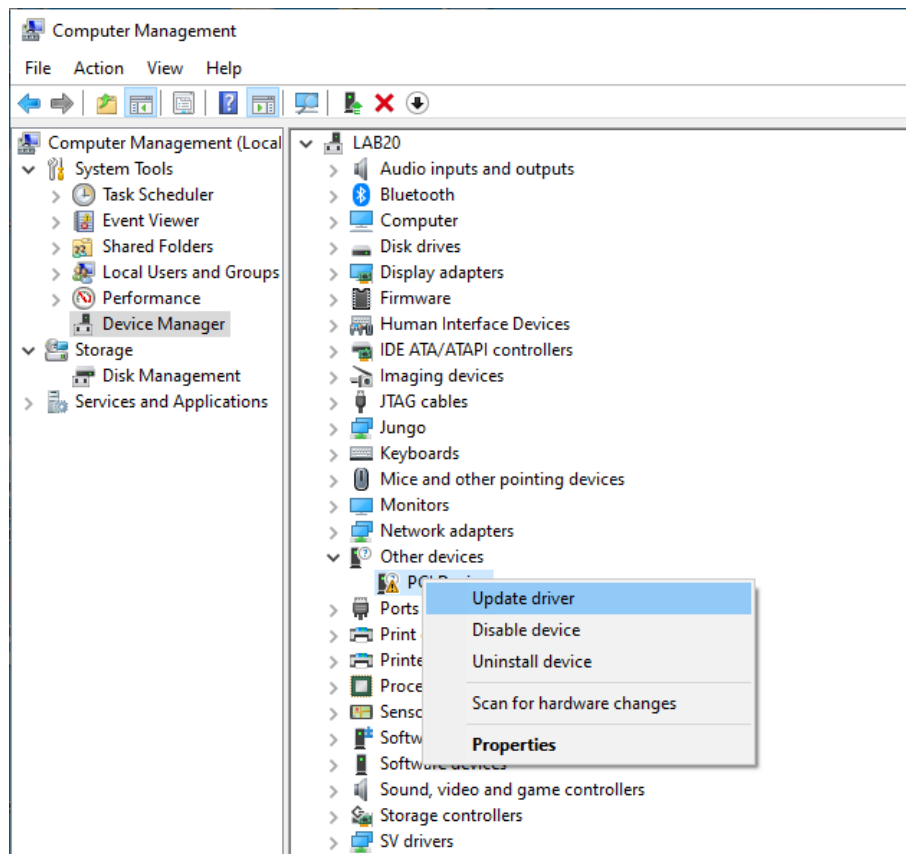


Figure 5-3 Screenshot of launching Update Driver Software... dialog

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 5-4**

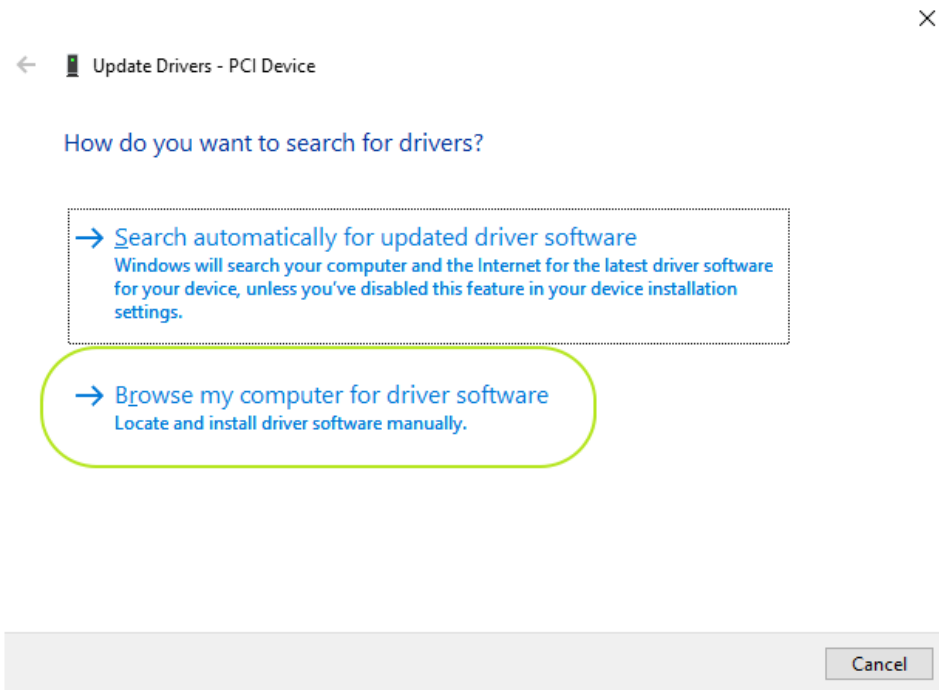


Figure 5-4 Dialog of Browse my computer for the driver software

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 5-5**. Click the **Next** button.

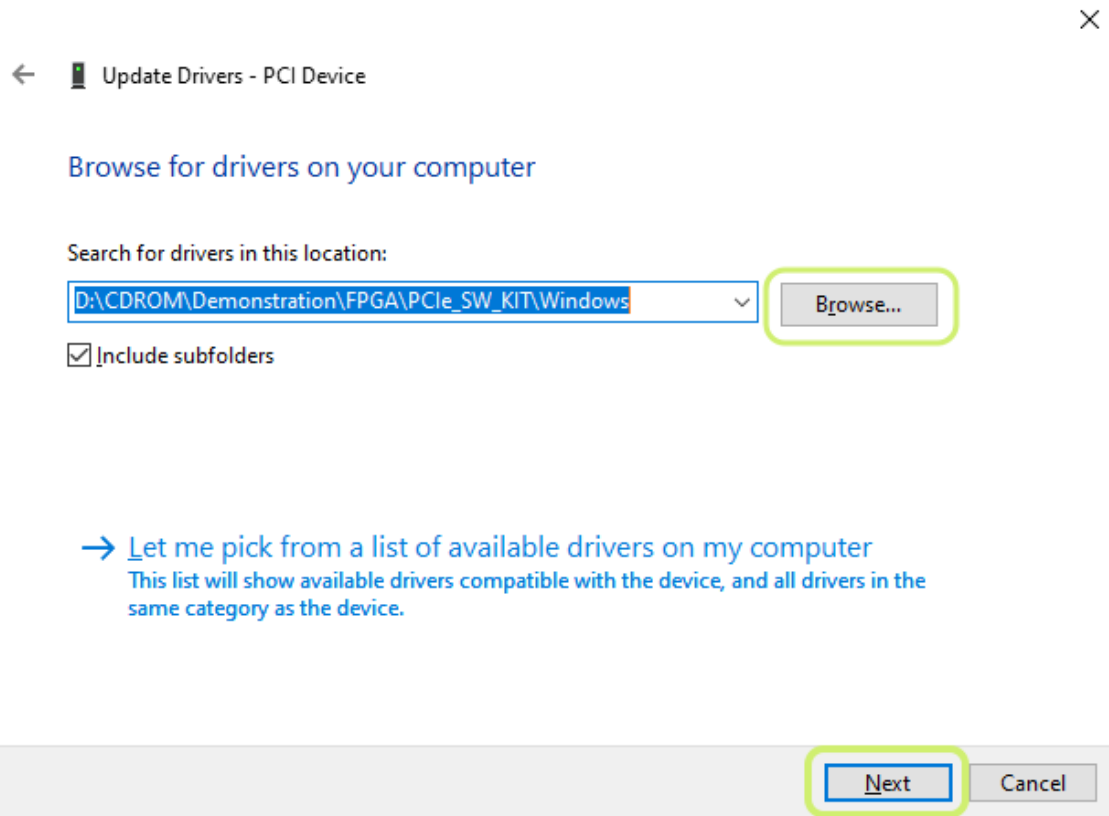


Figure 5-5 Browse for the driver software on your computer

9. When the **Windows Security** dialog appears, as shown **Figure 5-6**, click the **Install** button.

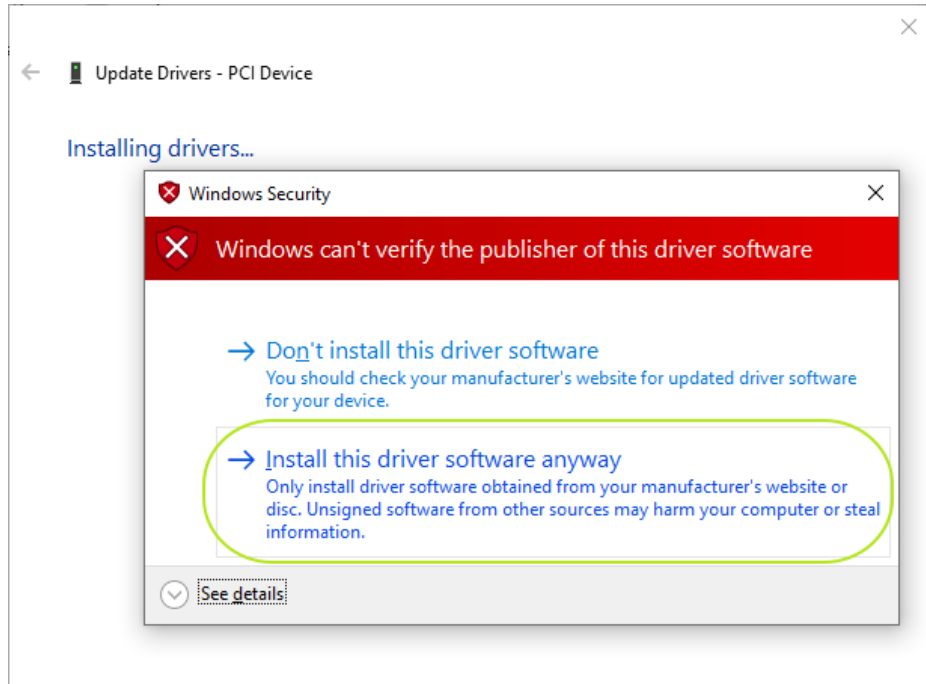


Figure 5-6 Click Install in the dialog of Windows Security

10. When the driver is installed successfully, the successfully dialog will appear, as shown in **Figure 5-7**. Click the **Close** button.

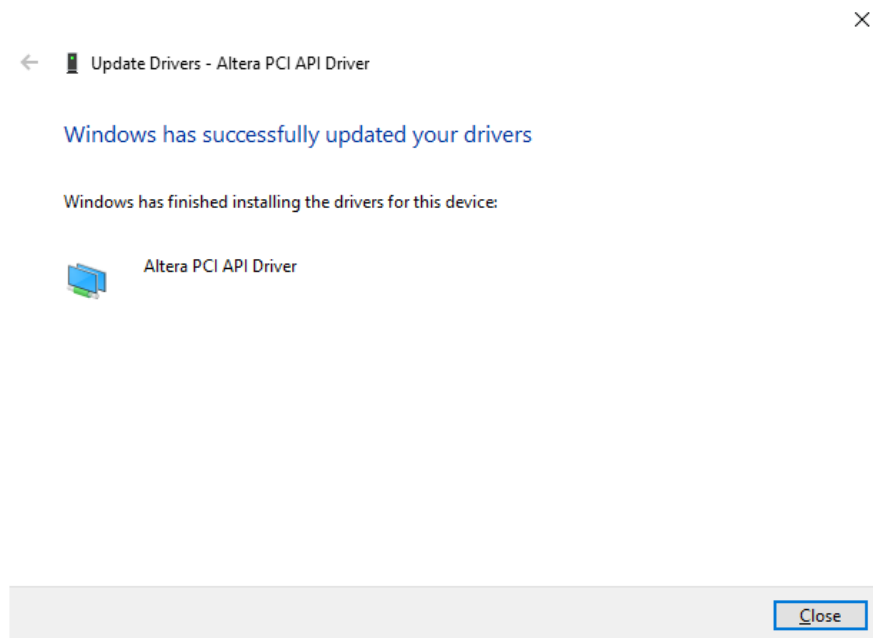


Figure 5-7 Click Close when the installation of the Altera PCI API Driver is complete

11. Once the driver is successfully installed, users can see the **Altera PCI API Driver** under the device manager window, as shown in **Figure 5-8**.

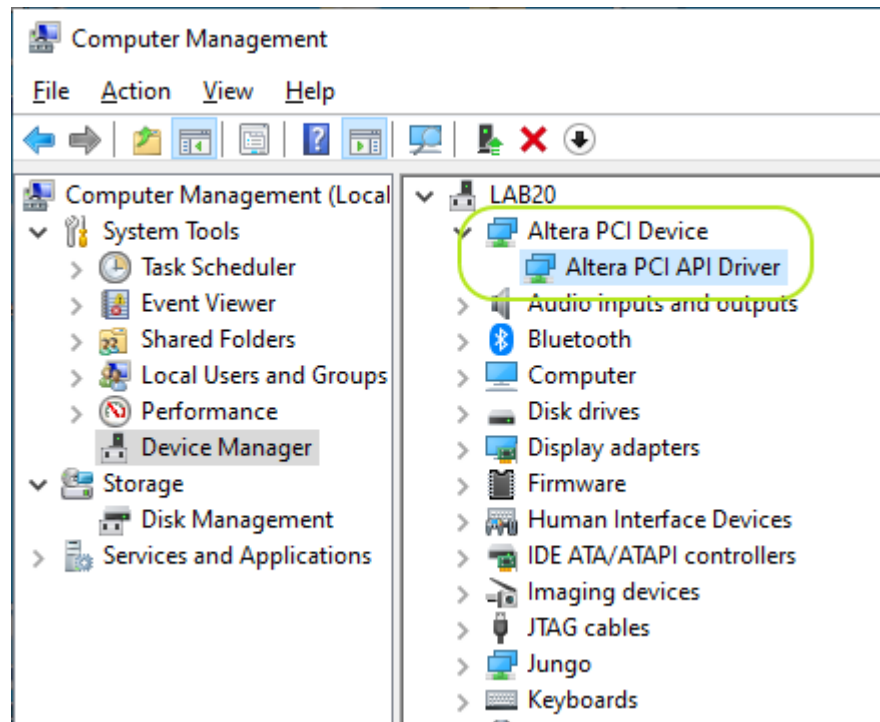


Figure 5-8 Altera PCI API Driver in Device Manager

■ Create a Software Application

All the files needed to create a PCIe software application are located in the directory `Resource_Package\Demonstration\FPGA\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 DDR4 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:

5.4 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:
<pre>PCIE_HANDLE PCIE_Open(Uint16_t wVendorID, Uint16_t wDeviceID, Uint16_t wSubVendorID, Uint16_t wSubDeviceID, Uint16_t wCardIndex);</pre>
Parameters:
<p>wVendorID: Specify the desired vendor ID. A zero value means to ignore the vendor ID.</p> <p>wDeviceID: Specify the desired device ID. A zero value means to ignore the device ID.</p> <p>wSubVendorID: Specify the desired subsystem vendor ID. A zero value means to ignore the subsystem vendor ID.</p> <p>wSubDeviceID: Specify the desired subsystem device ID. A zero value means to ignore the subsystem device ID.</p> <p>wCardIndex: Specify the matched card index, a zero based index, based on the matched vendor ID, device ID, subsystem vendor ID and subsystem device ID.</p>
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:
<pre>bool PCIE_ConfigRead32 (PCIE_HANDLE hPCIE, uint32_t Offset, uint32_t *pdwData);</pre>
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.

5.5 PCIe Reference Design - DDR4

The application reference design shows how to implement basic control and data transfer in DMA. In the design, basic I/O is used to control the BUTTON and LED on the FPGA board, and high-speed data transfer is performed by the DMA. Furthermore, it demonstrates how to add DDR4 Memory Controllers for the DDR4-A Component and DDR4-B Component, perform 4GB data DMA for both DDR4 Component, and call the “PCIE_ConfigRead32” API to check the PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

Resource_Package\Demonstration\FPGA\PCIe_DDR4\demo_batch

The folder includes following files:

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

■ Demonstration Setup

1. Install the FPGA board on your PC as shown in **Figure 5-9**.



Figure 5-9 FPGA board installation on PC

2. Configure the FPGA with the golden_top.sof by executing the test.bat.
3. Restart Windows
4. Install the PCIe driver if necessary. The driver is located in the folder:

Resource_Package\Demonstration\FPGA\PCIe_SW_KIT\Windows\PCIe_Driver.

5. Make sure that Windows has detected the FPGA Board by checking the Windows Device Manager as shown in **Figure 5-10**.

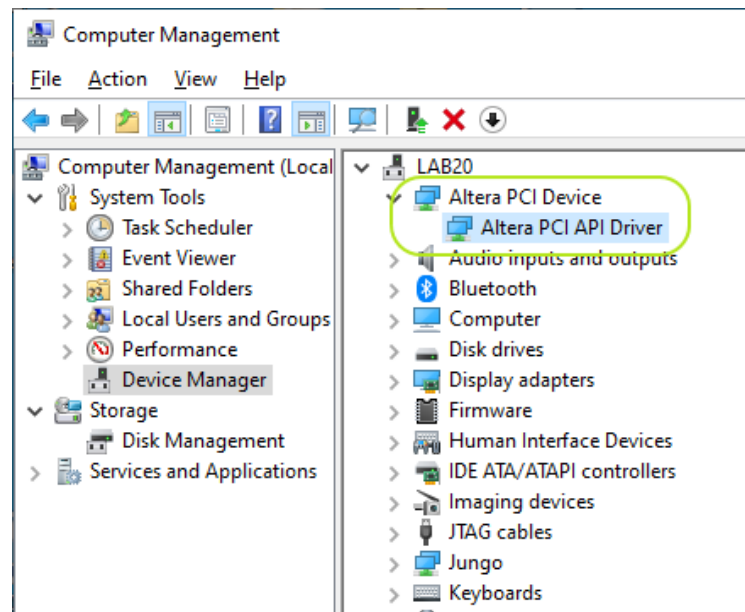


Figure 5-10 Screenshot for PCIe Driver

6. Go to windows_app folder, execute PCIE_DDR4.exe. A menu will appear as shown in **Figure 5-11**.

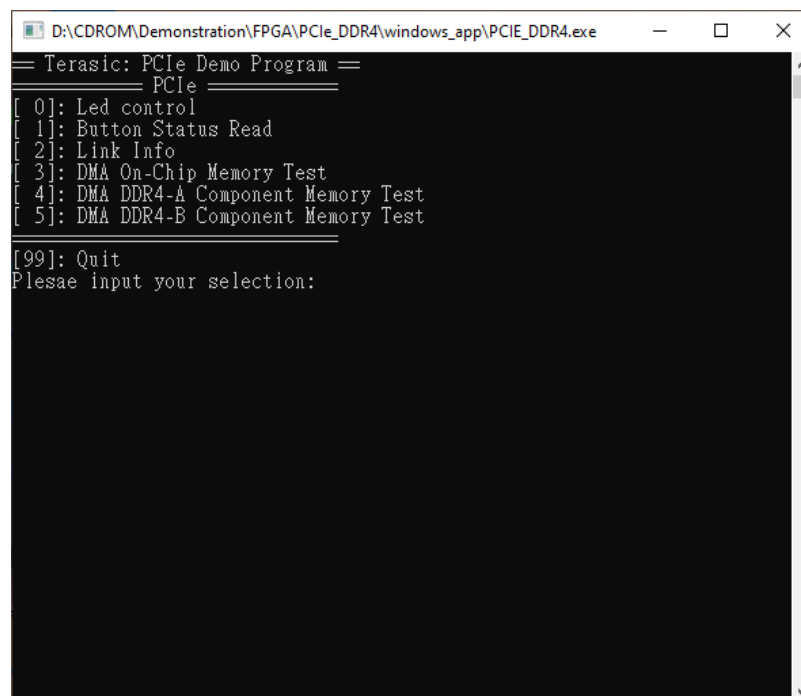
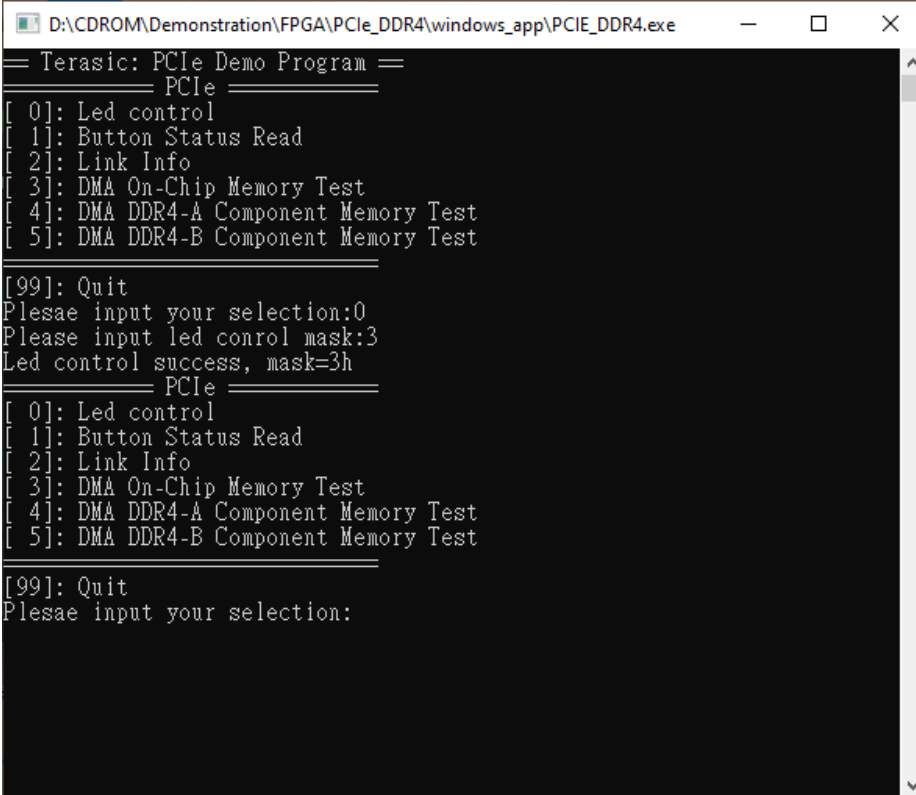


Figure 5-11 Screenshot of Program Menu

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 5-12**. If input 0 (hex 0x00), all LEDs will be turned off.



```
D:\CDROM\Demonstration\FPGA\PCIE_DDR4\windows_app\PCIE_DDR4.exe
== Terasic: PCIe Demo Program ==
== PCIe ==
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test

[99]: Quit
Plesae input your selection:0
Please input led control mask:3
Led control success, mask=3h
== PCIe ==
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test

[99]: Quit
Plesae input your selection:
```

Figure 5-12 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 5-13**.

```

D:\CDROM\Demonstration\FPGA\PCIe_DDR4\windows_app\PCIe_DDR4.exe
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:0
Please input led control mask:3
Led control success, mask=3h
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:1
Button status mask:=0h
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:

```

Figure 5-13 Screenshot of Button Status Report

9. Type 2 followed by the ENTER key to select the Link Info item. The PCIe link information will be shown as in **Figure 5-14**. Gen3 link speed and x16 link width are expected.

Note: The SOM-FMC edition supports a Link Width of "x8" as PCIe[15:12] lanes are only present in the SOM-FMC+ edition.

```
D:\CDROM\Demonstration\FPGA\PCIE_DDR4\windows_app\PCIE_DDR4.exe
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:1
Button status mask:=0h
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:2
Vender ID:1172h
Device ID:09C4h
Current Link Speed is Gen3
Negotiated Link Width is x16
Maximum Payload Size is 256-byte
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 5-14 Screenshot of Link Info

10. 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 5-15**.

```
D:\CDROM\Demonstration\FPGA\PCIE_DDR4\windows_app\PCIE_DDR4.exe
Device ID:09C4h
Current Link Speed is Gen3
Negotiated Link Width is x16
Maximum Payload Size is 256-byte
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Please 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
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Please input your selection:
```

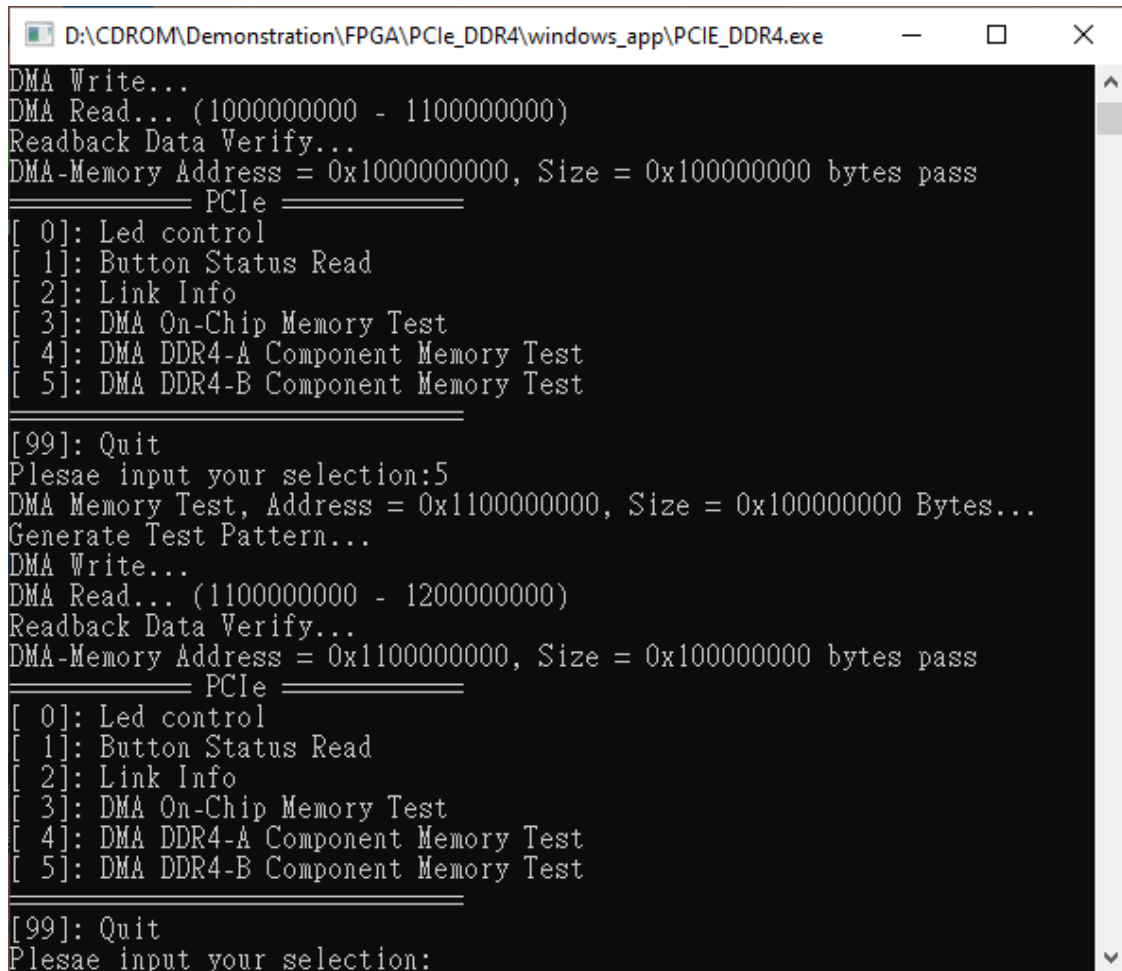
Figure 5-15 Screenshot of On-Chip Memory DMA Test Result

11. Type 4 followed by the ENTER key to select the DMA DDR4-A Component Memory Test item. The DMA write and read test result will be reported as shown in [Figure 5-16](#).


```
D:\CDROM\Demonstration\FPGA\PCIE_DDR4\windows_app\PCIE_DDR4.exe
DMA Write...
DMA Read... (100000 - 180000)
Readback Data Verify...
DMA-Memory Address = 0x100000, Size = 0x80000 bytes pass
==== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
====
[99]: Quit
Plesae input your selection:4
DMA Memory Test, Address = 0x1000000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read... (1000000000 - 1100000000)
Readback Data Verify...
DMA-Memory Address = 0x1000000000, Size = 0x100000000 bytes pass
==== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
====
[99]: Quit
Plesae input your selection:
```

Figure 5-16 Screenshot of the DDR4-A Component Memory DMA Test Result

12. 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 5-17](#).



```
D:\CDROM\Demonstration\FPGA\PCIE_DDR4\windows_app\PCIE_DDR4.exe
DMA Write...
DMA Read... (1000000000 - 1100000000)
Readback Data Verify...
DMA-Memory Address = 0x1000000000, Size = 0x100000000 bytes pass
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:5
DMA Memory Test, Address = 0x1100000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read... (1100000000 - 1200000000)
Readback Data Verify...
DMA-Memory Address = 0x1100000000, Size = 0x100000000 bytes pass
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 5-17 Screenshot of the DDR4-B Component Memory DMA Test Result

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

■ Development Tools

- Quartus Prime 24.3 Pro Edition
- Visual C++ 2019

■ Demonstration Source Code Location

- Quartus Project: Demonstration\FPGA\PCIE_DDR4
- Visual C++ Project: Demonstration\FPGA\PCIE_SW_KIT\Windows\PCIE_DDR4

■ FPGA Application Design

Figure 5-18 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.

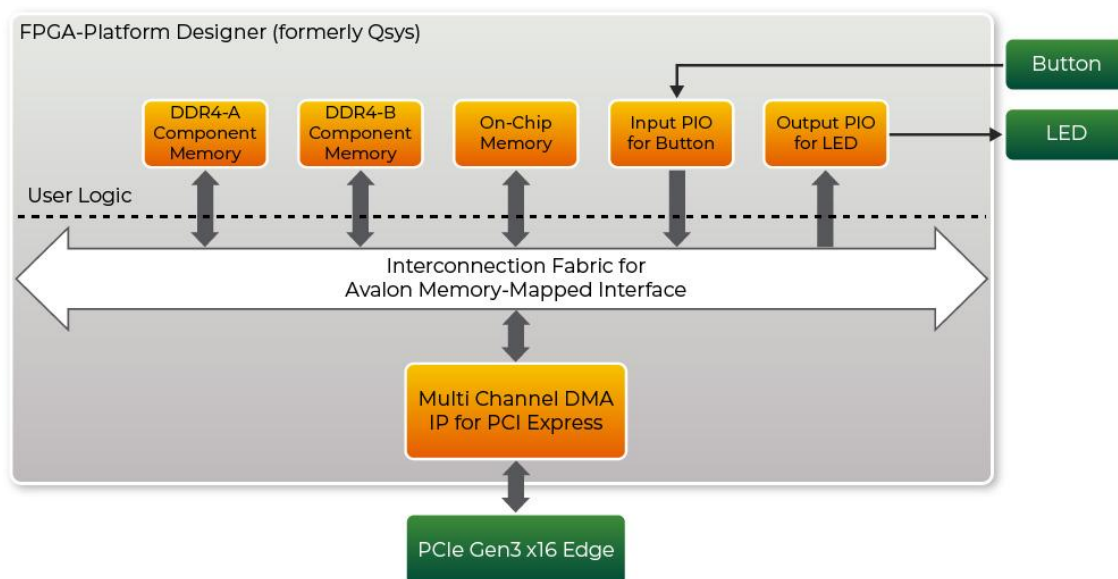


Figure 5-18 Hardware block diagram of the PCIe_DDR4 reference design

■ 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     0x1100000000

#define ONCHIP_MEM_TEST_SIZE         (512*1024) //512KB
#define DDR4A_MEM_TEST_SIZE          (4ull*1024*1024*1024) //4GB
#define DDR4B_MEM_TEST_SIZE          (4ull*1024*1024*1024) //4GB

```

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.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;
        case 5:
            printf("Current Link Speed is Gen5\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

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 Altera FPGA IP for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel document [ug20297-683821-844753](https://www.intel.com/content/dam/develop/external/us/en/documents/ug20297-683821-844753.pdf).

6.1 PCI Express System Infrastructure

Figure 6-1 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 Altera FPGA IP for PCI Express. The application software on the PC side is developed by Terasic based on Intel's PCIe kernel mode driver.

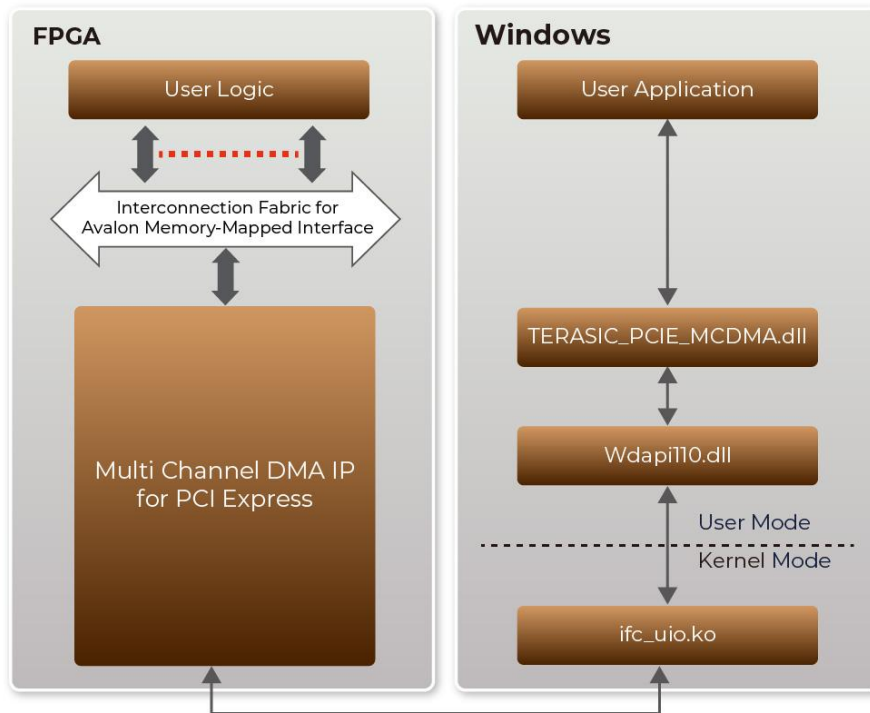


Figure 6-1 Infrastructure of PCI Express System

6.2 PC PCI Express Software SDK

The FPGA Resource_Package 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 “Resource_Package/Demonstration/FPGA/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.

6.3 Set the Boot Parameters

Follow the below step to modify the default hugepages setting in grub files:

Edit /etc/default/grub file.

Append the highlighted parameters in GRUB_CMDLINE_LINUX line in /etc/default/grub file.

```
GRUB_CMDLINE_LINUX=" rd.lvm.lv=centos/root\  
rd.lvm.lv=centos/swap rhgb default_hugepagesz=1G hugepagesz=1G\  
hugepages=5 panic=1 iommu=pt
```

1. For ubuntu 20.04:

After editing the /etc/default/grub

Please update grub.cfg by "**sudo update-grub2**"

2. For Centos:

Generate GRUB configuration files:

To check whether the boot system is legacy or EFI based, use the following file.

\$ls -al /sys/firmware/efi

If this file is present, then the boot system is EFI based; otherwise, legacy:

- If it is a legacy system, execute the following command:

\$ grub2-mkconfig -o /boot/grub2/grub.cfg

- If it is an EFI based system, execute the following command:

\$ grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg

Reference: <https://www.intel.com/content/www/us/en/docs/programmable/683517/24-3/set-the-boot-parameters-45392.html>

6.4 PCI Express Software Stack

Figure 6-2 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_uio.ko` kernel driver is provided by Intel.

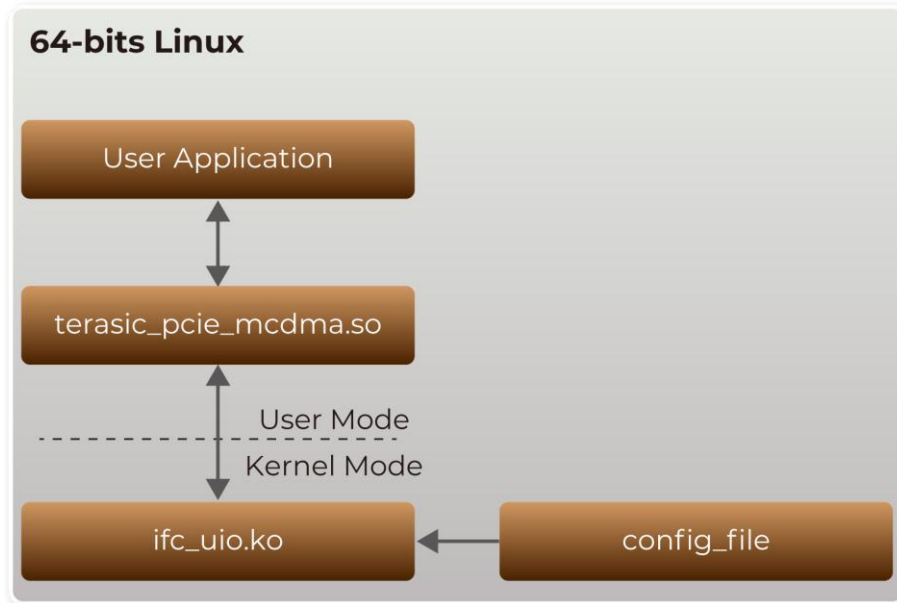


Figure 6-2 PCI Express Software Stack

■ Install PCI Express Driver on Linux

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

"Resource_Package/Demonstration/FPGA/PCIe_SW_KIT/Linux/PCIe_Driver"

The folder includes the following files:

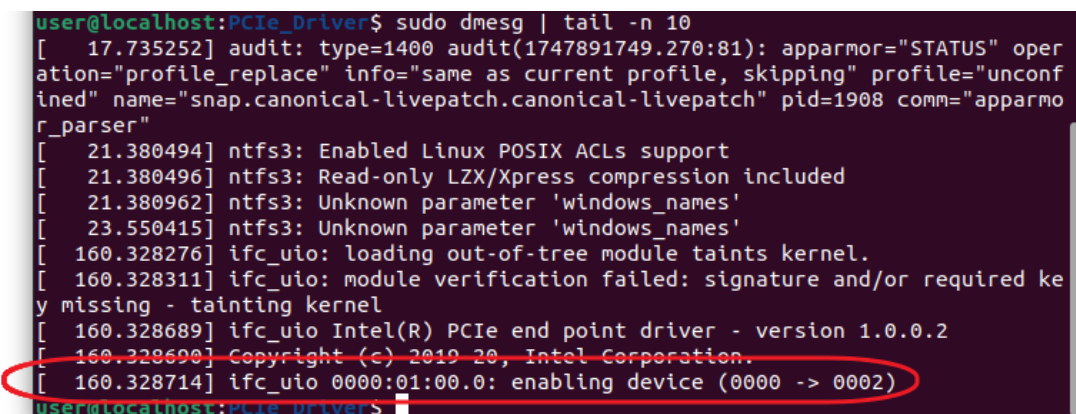
- `ifc_pci_uio.c`
- `ifc_pci_uio.h`
- `common/include/regs/pio_reg_registers.h`
- `common/include/regs/qdma_regs_2_registers.h`
- `common/include/ifc_mcdma.h`
- `common/include/ifc_mcdma_utils.h`
- `common/include/mcdma_ip_params.h`

- 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 "Resource_Package/Demonstration/FPGA/PCIe_DDR4/demo_batch".
4. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by typing the following commands in terminal. Replace "/home/user/intelFPGA_pro/24.3/quartus" to your quartus installation path.

`export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/24.3/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_uio.ko, and make sure driver is loaded successfully and FPGA is detected by the driver as shown in **Figure 6-3**.
 - make
 - sudo sh load_driver
 - dmesg | tail -n 10



```

user@localhost:PCIe_Driver$ sudo dmesg | tail -n 10
[ 17.735252] audit: type=1400 audit(1747891749.270:81): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unconfined" name="snap.canonical-livepatch.canonical-livepatch" pid=1908 comm="apparmor_parser"
[ 21.380494] ntfs3: Enabled Linux POSIX ACLs support
[ 21.380496] ntfs3: Read-only LZX/Xpress compression included
[ 21.380962] ntfs3: Unknown parameter 'windows_names'
[ 23.550415] ntfs3: Unknown parameter 'windows_names'
[ 160.328276] ifc_uio: loading out-of-tree module taints kernel.
[ 160.328311] ifc_uio: module verification failed: signature and/or required key missing - tainting kernel
[ 160.328689] ifc_uio Intel(R) PCIe end point driver - version 1.0.0.2
[ 160.328690] Copyright (c) 2019-20, Intel Corporation.
[ 160.328714] ifc_uio 0000:01:00.0: enabling device (0000 -> 0002)
user@localhost:PCIe_Driver$
  
```

Figure 6-3 Screenshot of install PCIe driver

■ Create a Software Application

All the files needed to create a PCIe software application are located in the directory `Resource_Package/Demonstration/FPGA/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 DDR4 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.

6.5 PCI Express Library API

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

6.6 PCIe Reference Design - DDR4

The application reference design shows how to implement basic control and data transfer in DMA. In the design, basic I/O is used to control the BUTTON and LED on the FPGA board, and high-speed data transfer is performed by the DMA. Furthermore, it demonstrates how to add DDR4 Memory Controllers for the DDR4-A Component and DDR4-B Component, perform 4GB data DMA for both DDR4 Component, and call the "PCIE_ConfigRead32" API to check the PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

Resource_Package/Demonstration/FPGA/PCle_DDR4/demo_batch

The folder includes following files:

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

■ Demonstration Setup

1. Install the FPGA board on your PC as shown in **Figure 6-4**.



Figure 6-4 FPGA board installation on PC

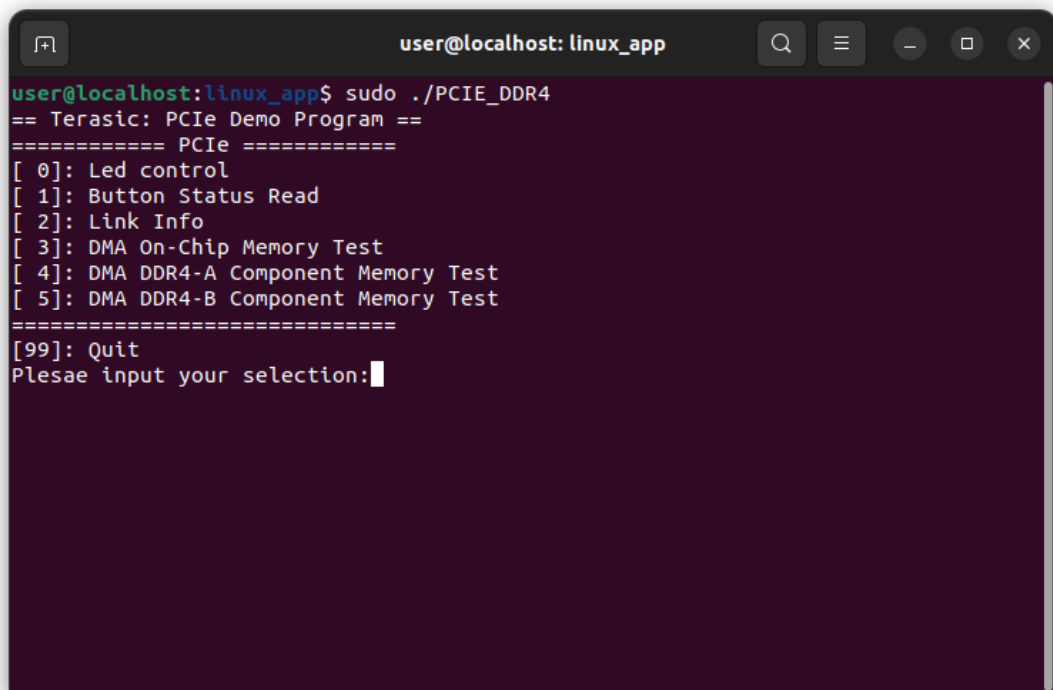
2. Open a terminal and use "cd" command to go to "Resource_Package/Demonstration/FPGA/PCIe_DDR4/demo_batch".
3. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by typing the following commands in the terminal. Replace /home/user/intelFPGA_pro/24.3/quartus to your Quartus installation path.

```
export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/24.3/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:
Resource_Package/Demonstration/FPGA/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.

```
user@localhost:PCIe_Driver$ lspci -nn | grep 1172:09c4
01:00.0 Unassigned class [ff00]: Altera Corporation Device [1172:09c4] (rev 01)
user@localhost:PCIe_Driver$
```

8. Go to the linux_app folder, execute "sudo ./PCIE_DDR4". A menu will appear as shown in **Figure 6-5**.



```
user@localhost:linux_app$ sudo ./PCIE_DDR4
== Terasic: PCIe Demo Program ==
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 6-5 Screenshot of Program Menu

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


```
user@localhost: linux_app
user@localhost:linux_app$ sudo ./PCIE_DDR4
== Terasic: PCIE Demo Program ==
===== PCie =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:0
Please input led conrol mask:3
Led control success, mask=3h
===== PCie =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 6-6 Screenshot of LED Control

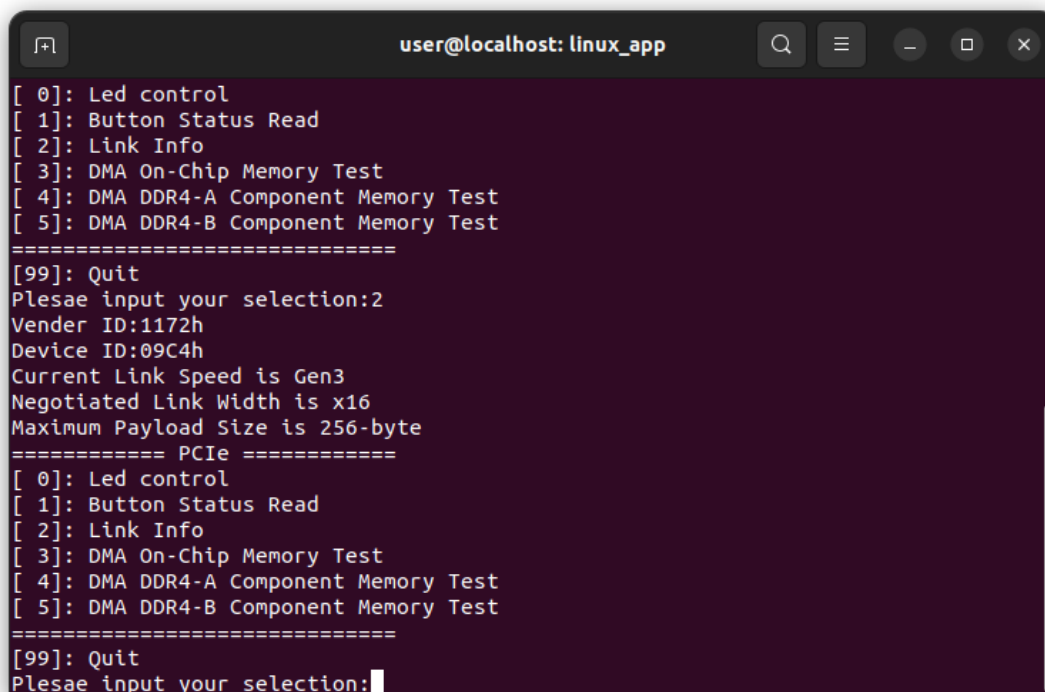
10. Type 1 followed by the ENTER key to select the Button Status Read item. The button status will be reported as shown in [Figure 6-7](#).

```
user@localhost: linux_app
Plesae input your selection:0
Please input led conrol mask:3
Led control success, mask=3h
===== PCie =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:1
Button status mask:=0h
===== PCie =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```


Figure 6-7 Screenshot of Button Status Report

11. Type 2 followed by the ENTER key to select the Link Info item. The PCIe link information will be shown as in **Figure 6-8**. Gen3 link speed and x16 link width are expected.

Note: The SOM-FMC edition supports a Link Width of "x8" as PCIe[15:12] lanes are only present in the SOM-FMC+ edition.



```
user@localhost: linux_app
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:2
Vender ID:1172h
Device ID:09C4h
Current Link Speed is Gen3
Negotiated Link Width is x16
Maximum Payload Size is 256-byte
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 6-8 Screenshot of Link Info

12. 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 6-9**.

```
user@localhost: linux_app
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component 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
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

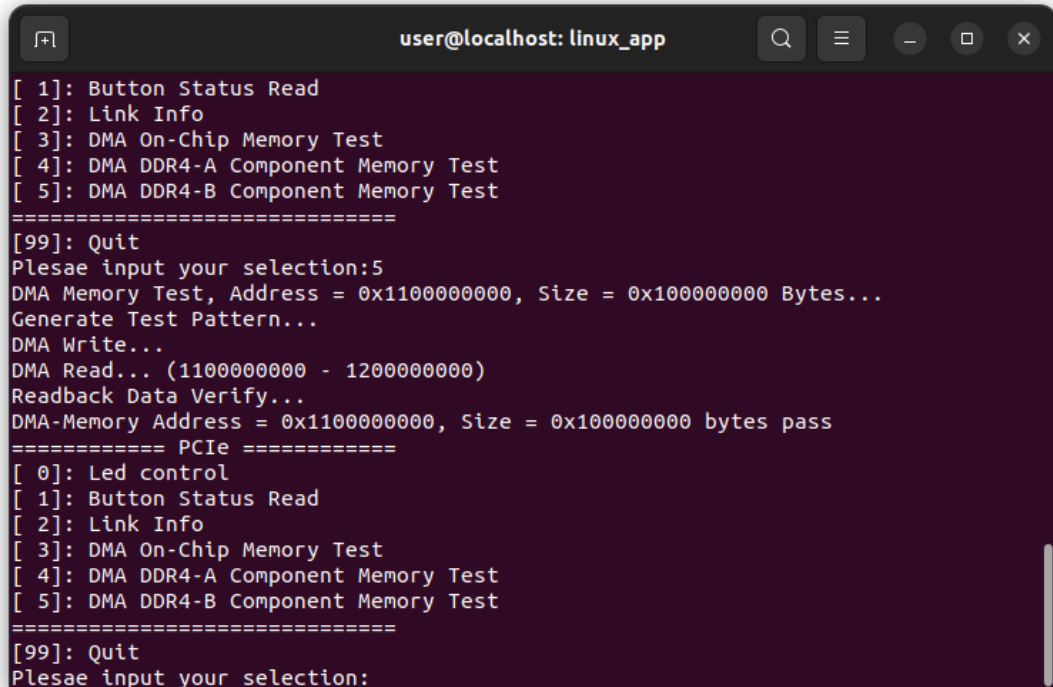
Figure 6-9 Screenshot of On-Chip Memory DMA Test Result

13. Type 4 followed by the ENTER key to select the DMA DDR4-A COMPONENT Memory Test item. The DMA write and read test result will be reported as shown in [Figure 6-10](#).

```
user@localhost: linux_app
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:4
DMA Memory Test, Address = 0x100000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read... (100000000 - 1100000000)
Readback Data Verify...
DMA-Memory Address = 0x100000000, Size = 0x100000000 bytes pass
===== PCIe =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 6-10 Screenshot of DDR4-A COMPONENT Memory DAM Test Result

14. 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 6-11](#).



```
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:5
DMA Memory Test, Address = 0x1100000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read... (1100000000 - 1200000000)
Readback Data Verify...
DMA-Memory Address = 0x1100000000, Size = 0x100000000 bytes pass
===== PCIE =====
[ 0]: Led control
[ 1]: Button Status Read
[ 2]: Link Info
[ 3]: DMA On-Chip Memory Test
[ 4]: DMA DDR4-A Component Memory Test
[ 5]: DMA DDR4-B Component Memory Test
=====
[99]: Quit
Plesae input your selection:
```

Figure 6-11 Screenshot of DDR4-B COMPONENT Memory DAM Test Result

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

■ Development Tools

- Quartus Prime 24.3 Pro Edition
- GNU Compiler Collection, Version 9.4 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstration/FPGA/PCIE_DDR4
- C++ Project: Demonstration/FPGA/PCie_SW_KIT/Linux/PCie_DDR4

■ FPGA Application Design

Figure 6-12 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.

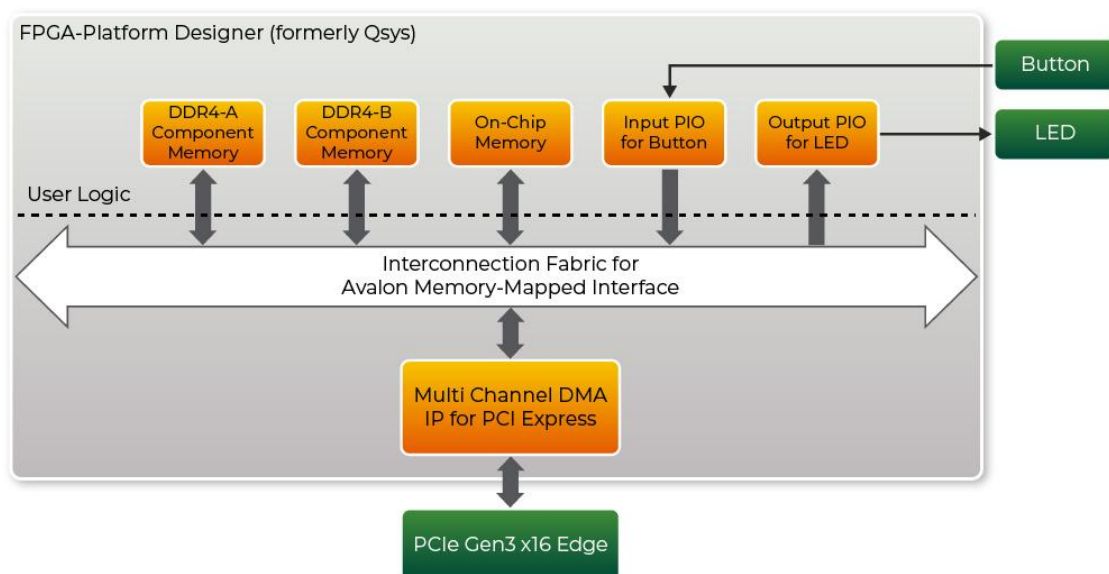


Figure 6-12 Hardware block diagram of the PCIe_DDR4 reference design

■ 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     0x1100000000

#define ONCHIP_MEM_TEST_SIZE         (512*1024) //512KB
#define DDR4A_MEM_TEST_SIZE          (4ull*1024*1024*1024) //4GB
#define DDR4B_MEM_TEST_SIZE          (4ull*1024*1024*1024) //4GB

```

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 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;
        case 5:
            printf("Current Link Speed is Gen5\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 7

MCIO DDR4

This will introduce how to use MCIO connectors to establish a PCIe connection with a PC and transfer data. This chapter will mainly introduce how to perform hardware connection and settings. For PCIe-related transmission settings and demonstration, please refer to chapter 5 and chapter 6.

7.1 Hardware Setting

This section will introduce how to connect the MICO connector of the FPGA board to the Host, as well as related power supply and download cable settings. As shown in **Figure 7-1**, please follow the steps below to set up the hardware connection.

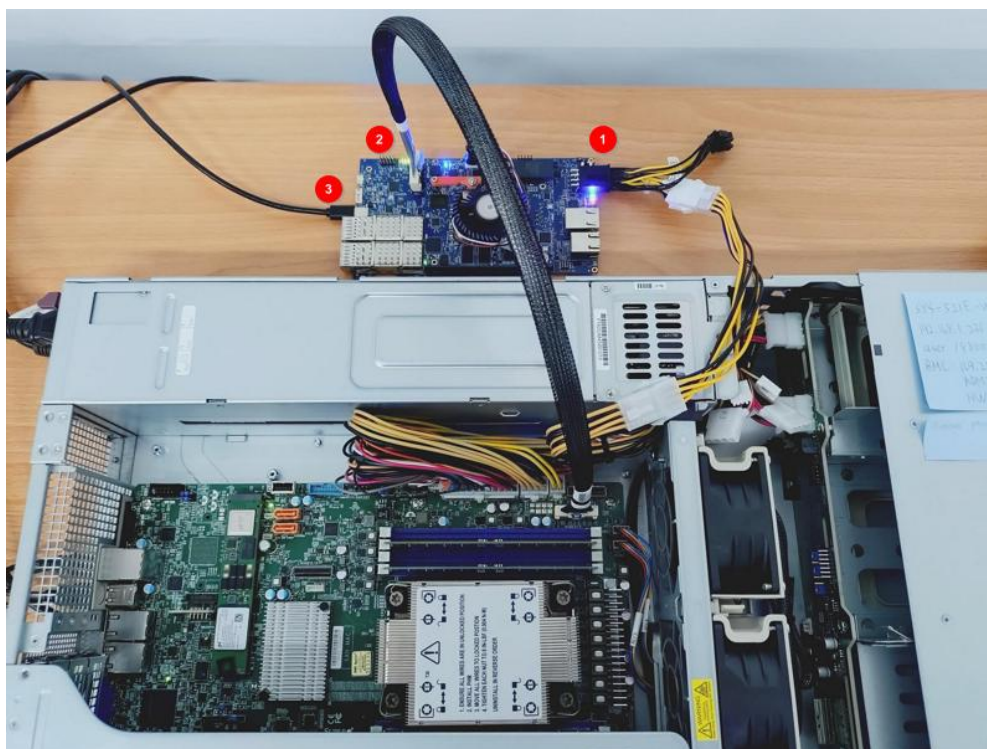


Figure 7-1 Hardware setup for MCIO demonstration

1. Plug 8-pin 12V ATX power to the 2x4 power connector of the board.
2. Using MCIO cable to connect the FPGA board and Host.
3. Connect Host and FPGA board by Micro USB cable to establish the USB blaster connection for programming the FPGA.

7.2 PCIe Reference Design – MCIO DDR4

The application reference design shows how to implement basic control and data transfer in DMA. In the design, basic I/O is used to control the BUTTON and LED on the FPGA board, and high-speed data transfer is performed by the DMA. Furthermore, it demonstrates how to add DDR4 Memory Controllers for the DDR4-A Component and DDR4-B Component, perform 4GB data DMA for both DDR4 Component, and call the “PCIE_ConfigRead32” API to check the PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

Resource_Package\Demonstration\FPGA\MCIO_DDR4\demo_batch

For Window OS:

The folder includes following files:

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

For Linux OS:

The folder includes following files:

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

■ Demonstration Setup

1. Please refer to section 6.1 to complete the hardware settings.
2. Configure FPGA with golden_top.sof by executing the test.bat or test.sh (Linux OS).
3. For subsequent PCIe driver installation and demo execution, please refer to section 4.5 (Windows OS) or section 5.6 (Linux OS).
4. For PCIe software SDK, software stack and library API, please refer to chapter 4 (Window OS) or chapter 5 (Linux OS).

Chapter 8

Transceiver Verification

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

8.1 Transceiver Test Code

The transceiver test code is used to verify the transceiver channels via the QSPF ports through an external loopback method. The transceiver channels are verified with the data rates 10.3125 Gbps with **NRZ** modulation.

- Loopback Fixture

To enable an external loopback of the transceiver channels, QSPF loopback fixtures (Molex 074730025) are required, as shown in **Figure 8-1**.



Figure 8-1 QSPF Loopback Fixture

Figure 8-2 shows the FPGA board with two QSPF loopback fixtures installed.



Figure 8-2 QSFP Transceiver Loopback Test in Progress

- Testing by Transceiver Test Code

The transceiver test code is available in the folder `Resource_Package\Tool\Transceiver_Test`.

Figure 8-3 and **Figure 8-4** shows the L-Tile/H-Tile Transceiver Native PHY settings in the test code. The data rate of each transceiver channel is set to 10312.5 Mbps and the PMA modulation type is NRZ. So the 40Gbps QSFP loopback test code is implemented (4 channels in total). Also, the ATX PLL setting is shown in **Figure 8-4**.

L-Tile/H-Tile Transceiver Native PHY Intel Stratix 10 FPGA IP
altera_xcivr_native_s10_htile

[Details](#)
[Generate Example Design...](#)

Design Environment
 This component supports multiple interface views:
 System

General
 Message level for rule violations: error
☐ Use fast reset for simulation

Common PMA Options
 VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_0V
 Transceiver Link Type: lr

Datapath Options
 Transceiver channel type: GX
 Transceiver configuration rules: Basic (Enhanced PCS)
 PMA configuration rules: basic
 Transceiver mode: TX/RX Duplex
 Number of data channels: 4
 Data rate: 10312.5 Mbps

Figure 8-3 The Transceiver Native PHY setting

TX PMA
RX PMA
Enhanced PCS
PCS-Core Interface
Analog PMA Settings
Dynamic Recon

TX Bonding Options
 TX channel bonding mode: Not bonded
 PCS TX channel bonding master: AUTO
 Actual PCS TX channel bonding master: 0
 PCS reset sequence: Independent

TX PLL Options
 TX local clock division factor: 1
 Number of TX PLL clock inputs per channel: 1
 Initial TX PLL clock input selection: 0
Note - The external TX PLL IP must be configured with an output clock frequency of 5156.25 MHz.

TX PMA Optional Ports
☐ Enable tx_pma_jdtxrx_clkout port
☐ Enable tx_pma_elecidle port

Figure 8-4 TX PMA Setting

RX CDR Options

Number of CDR reference clocks: 1

Selected CDR reference clock: 0

Selected CDR reference clock frequency: 644.531250 MHz

PPM detector threshold: 1000 PPM

RX PMA Optional Ports

☐ Enable rx_pma_iqtxrx_clkout port

☐ Enable rx_pma_clkslip port

☒ Enable rx_is_lockedto data port

☒ Enable rx_is_lockedto ref port

☐ Enable rx_set_lockto data and rx_set_lockto ref ports

☐ Enable PRBS verifier control and status ports

☐ Enable rx_serialpbken port

Figure 8-5 RX PMA Setting

L-Tile/H-Tile Transceiver ATX PLL Intel Stratix 10 FPGA IP
altera_xcvt_atx_pll_s10_h_tile

General

Message level for rule violations: error

Protocol mode: Basic

Bandwidth: high

Number of PLL reference clocks: 1

Selected reference clock source: 0

VCCR_GXB and VCCT_GXB supply voltage for the Transceiver: 1_0V

Note - All PLLs and Native PHY instances in a given tile must be configured with the same supply voltage

Ports

GXT Configuration Options

Output Frequency

PLL output frequency: 5156.25 MHz

PLL output data rate: 10312.5 Mbps

PLL auto mode reference clock frequency (Integer): 644.53125 MHz

Figure 8-6 The ATX PLL setting

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

Direction	Item	Value
TX	VOD	31
	PostTap	-3
	PreTap	-6

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 QSFP loopback fixtures.
4. Connect your FPGA board to your PC with a type-C USB cable.
5. Power on the FPGA board
6. Execute 'test.bat" in the Transceiver_Test folder under your local disk.
7. The batch file will download .sof and .elf files, and start the test immediately. The test result is shown in the command line window, as shown in **Figure 8-7**. Note, In the SOM-FMC edition, QSFP_A shows “NoSync” because the XCVR lanes are only available in the SOM-FMC+ edition.
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 8-8**.

```

C:\WINDOWS\system32\cmd. X + v
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.

Program received signal SIGINT, Interrupt.
0x00000004 in ?? ()
OK. Software reset asserted
Loading section .entry, size 0x20 lma 0x0
Loading section .exceptions, size 0x2f4 lma 0x20
Loading section .text, size 0x2891c lma 0x314
Loading section .rodata, size 0x2e70 lma 0x28c30
Loading section .rwddata, size 0x1e2c lma 0x2d8cc
Start address 0x000057c0, load size 186572
Transfer rate: 375 KB/sec, 26653 bytes/write.
[Inferior 1 (Remote target) detached]
Transceiver for QSFP A/B testing...
Press buttons on the board can terminate the testing.
Apply default setting...done
===== Time Elapsed: 0 Seconds =====
QSFP_A 10 Gpbs-0: PASS, XferCnt:3019702272(0.35GB)
QSFP_A 10 Gpbs-1: PASS, XferCnt:3018858496(0.35GB)
QSFP_A 10 Gpbs-2: PASS, XferCnt:3018006528(0.35GB)
QSFP_A 10 Gpbs-3: PASS, XferCnt:3014631424(0.35GB)
QSFP_B 10 Gpbs-0: PASS, XferCnt:3014541312(0.35GB)
QSFP_B 10 Gpbs-1: PASS, XferCnt:3009929216(0.35GB)
QSFP_B 10 Gpbs-2: PASS, XferCnt:3010945024(0.35GB)
QSFP_B 10 Gpbs-3: PASS, XferCnt:3009724416(0.35GB)

```

Figure 8-7 QSFP A/B Transceiver Loopback Test in Progress

```
C:\WINDOWS\system32\cmd. X + v
QSFP_B 10 Gpbs-3: PASS, XferCnt:313187401728(36.46GB)
===== Time Elapsed: 35 Seconds =====
QSFP_A 10 Gpbs-0: PASS, XferCnt:364894265344(42.48GB)
QSFP_A 10 Gpbs-1: PASS, XferCnt:364893421568(42.48GB)
QSFP_A 10 Gpbs-2: PASS, XferCnt:364892569600(42.48GB)
QSFP_A 10 Gpbs-3: PASS, XferCnt:364889194496(42.48GB)
QSFP_B 10 Gpbs-0: PASS, XferCnt:364889473024(42.48GB)
QSFP_B 10 Gpbs-1: PASS, XferCnt:364884852736(42.48GB)
QSFP_B 10 Gpbs-2: PASS, XferCnt:364885876736(42.48GB)
QSFP_B 10 Gpbs-3: PASS, XferCnt:364884656128(42.48GB)
user abort!
stop xcvr...
disable PRBS...
Transceiver Testing is terminated.
|
```

Figure 8-8 QSFP Transceiver Loopback is terminated

Chapter 9

Additional Information

9.1 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

Titan S10 Evaluation Kit Web: de25-standard.terasic.com

■ Revision History

Date	Version	Changes
2025.10	V1.0	