TR5-Lite

FPGA Development Kit
PCIe Qsys Example Designs
User Manual

Stratix V | Flash | PCI EXPRESS | DDR3 | QDRII+ | SFP+ | SERIAL ATA

terasIC
www.terasic.com

Contents

Chapter 1

PCI Express 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 and FPGA communicate with each other through the PCI Express interface. V-Series Avalon-MM DMA for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel FPGA document: ug_pcie_avmm_dma.pdf.

1.1 PCI Express System Infrastructure

Figure 1-1 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 V-Series Avalon-MM DMA for PCI Express. The application software on the PC side is developed by Terasic based on Altera's PCIe kernel mode driver.

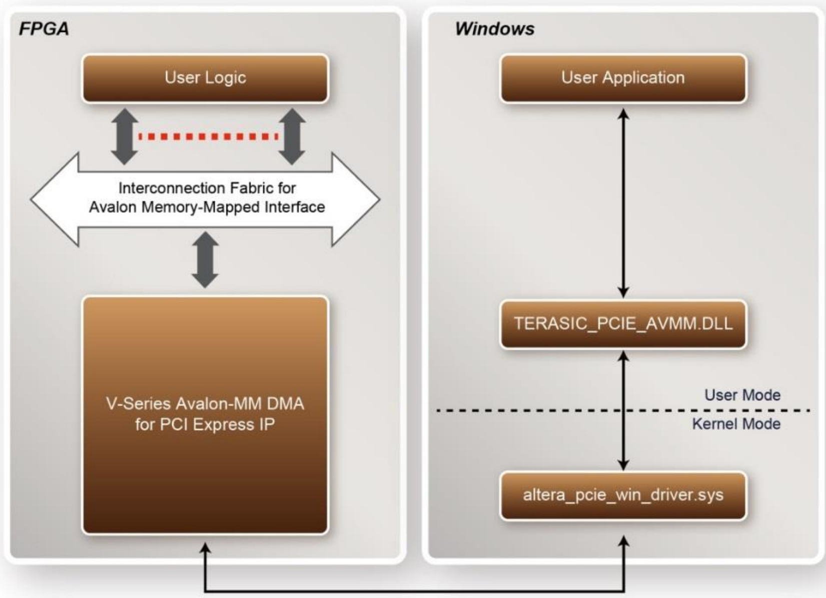


Figure 1-1 Infrastructure of PCI Express System

1.2 PC PCI Express Software SDK

The FPGA System CD contains a PC Windows based SDK to allow users to develop their 64-bit software application on 64-bits Windows XP/7/10. The SDK is located in the "CDROM\Demonstrations\PCIe_SW_KIT\Windows" folder which includes:

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

The kernel mode driver assumes the PCIe vender ID (VID) is 0x1172 and the device ID (DID) is 0xE003. If different VID and DID are used in the design, users need to modify the PCIe vender ID (VID) and device ID (DID) in the driver INF file accordingly.

The PCI Express Library is implemented as a single DLL called TERASIC_PCIE_AVMM.DLL. This file is a 64-bits DLL. With the DLL 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, DMA is required as the read and write operations are specified under the hardware design on the FPGA.

1.3 PCI Express Software Stack

Figure 1-2 shows the software stack for the PCI Express application software on 64-bit

Windows. The PCIe library module TERASIC_PCIE_AVMM.DLL provides DMA and direct I/O access for 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.

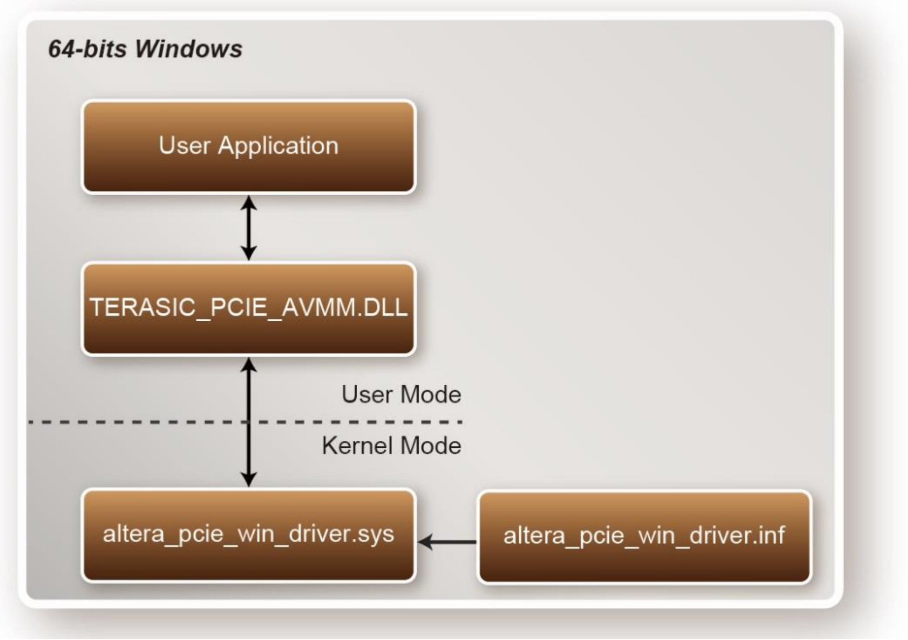


Figure 1-2 PCI Express Software Stack

Install PCI Express Driver on Windows

The PCIe driver is locate in the folder:

"CDROM\Demonstrations\PCIe_SW_KIT\Windows\PCIe_Driver"

The folder includes the following four files:

- Altera_pcie_win_driver.cat
- Altera_pcie_win_driver.inf
- Altera_pcie_win_driver.sys
- WdfCoinstaller01011.DLL

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

1. Install the TR5-Lite on the PCIe slot of the host PC
2. Make sure Altera Programmer and USB-Blaster II driver are installed
3. Execute test.bat in "CDROM\Demonstrations\PCIe_Fundamental\demo_batch" to configure the FPGA
4. Restart windows operation system
5. Click Control Panel menu from Windows Start menu. Click Hardware and Sound item before clicking the Device Manager to launch the Device Manager dialog. There will be a PCI Device item in the dialog, as shown in **Figure 1-3**. Move the mouse cursor to the PCI Device item and right click it to select the Update Driver Software... item.

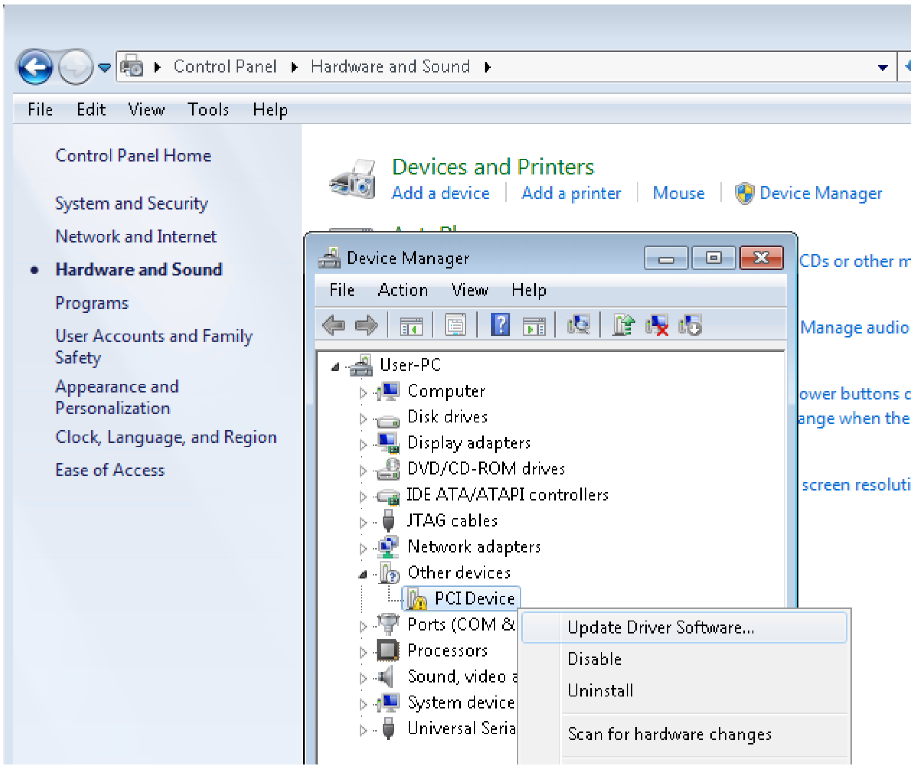


Figure 1-3 Screenshot of launching Update Driver Software… dialog

6. In the **How do you want to search for driver software** dialog, click **Browse my computer for driver software** item, as shown in **Figure 1-4**

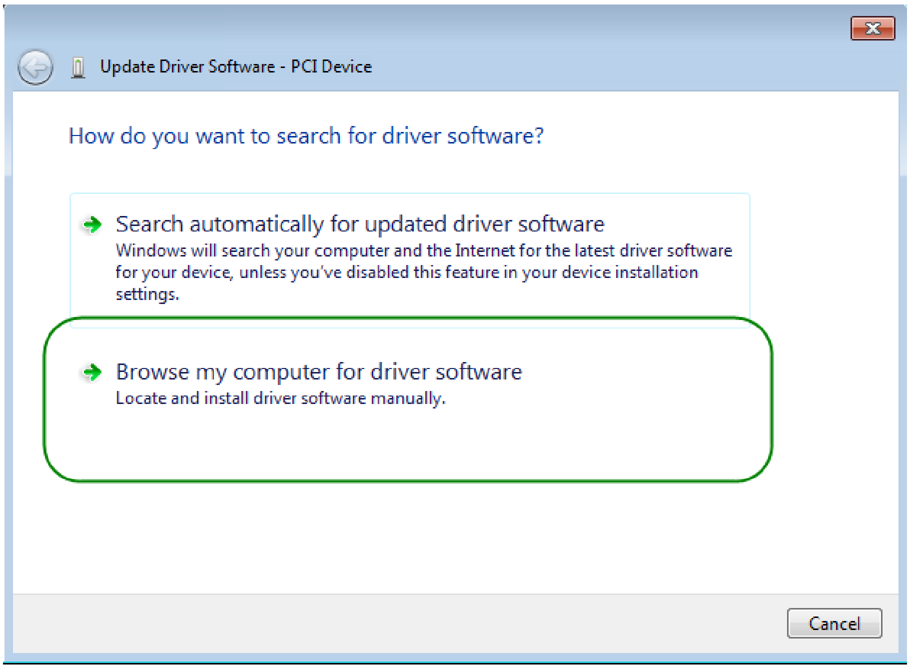


Figure 1-4 Dialog of Browse my computer for driver software

7. In the **Browse for driver software on your computer** dialog, click the **Browse** button to specify the folder where altera_pcie_win_driver.inf is located, as shown in **Figure 1-5**. Click the **Next** button.

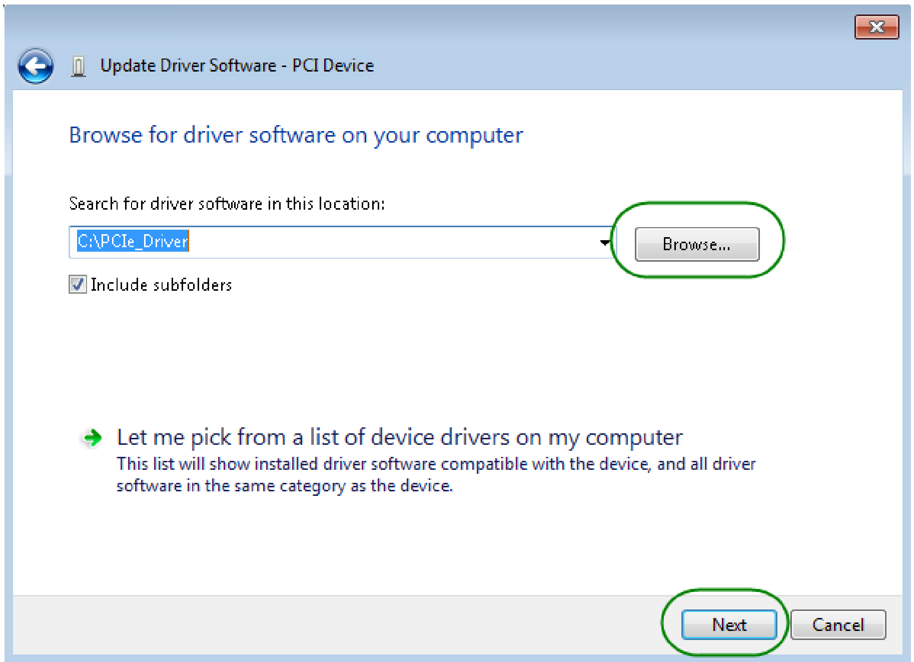


Figure 1-5 Browse for driver software on your computer

8. When the **Windows Security** dialog appears, as shown **Figure 1-6**, click the **Install** button.

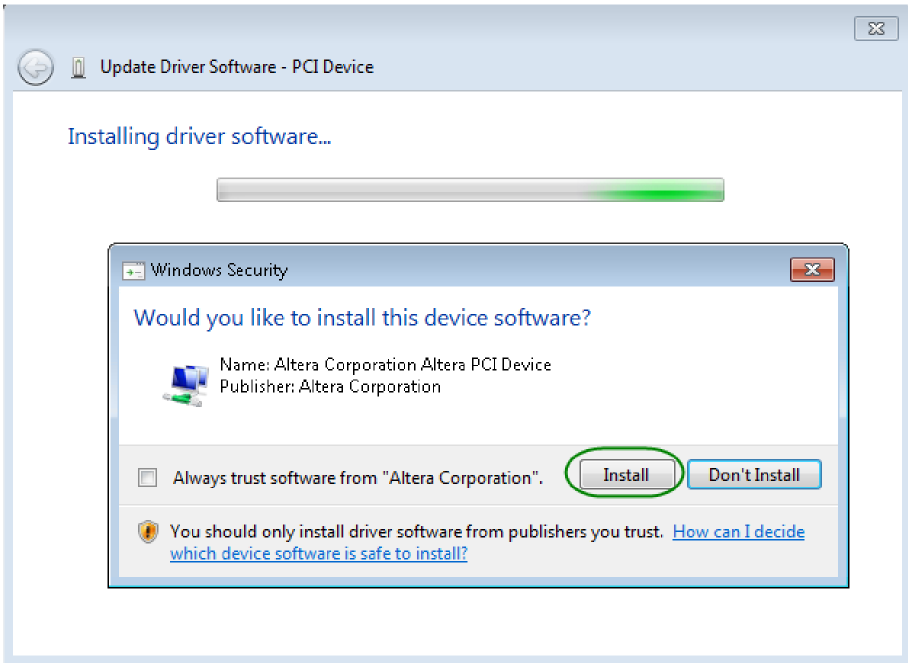


Figure 1-6 Click Install in the dialog of Windows Security

9. When the driver is installed successfully, the successfully dialog will appears, as shown in **Figure 1-7**. Click the **Close** button.

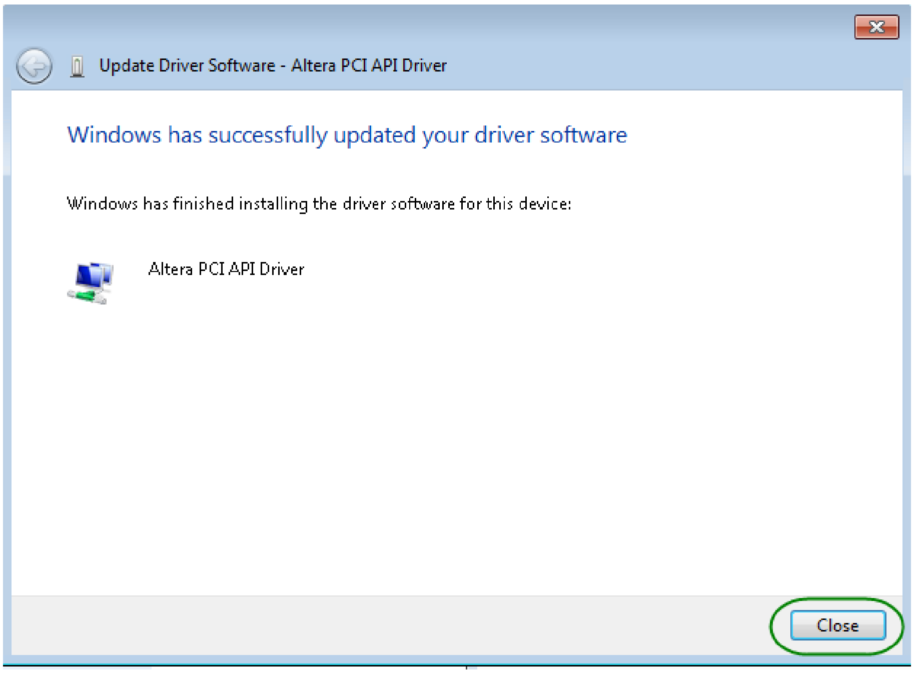


Figure 1-7 Click Close when the installation of Altera PCI API Driver is complete

10. Once the driver is successfully installed, users can see the **Altera PCI API Driver** under the device manager window, as shown in **Figure 1-8**.

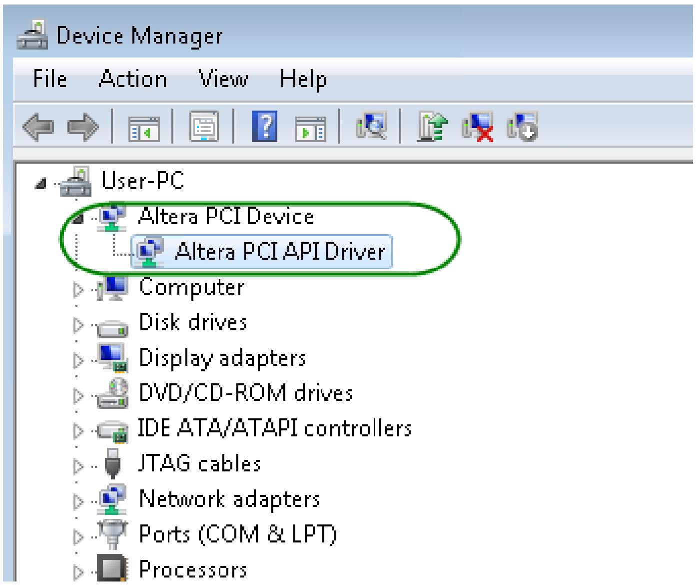


Figure 1-8 Altera PCI API Driver in Device Manager

■ Create a Software Application

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

- TERASIC_PCIE_AVMM.h
- TERASIC_PCIE_AVMM.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_AVMM.h in the C/C++ project.
3. Copy TERASIC_PCIE_AVMM.DLL to the folder where the project.exe is located.
4. Dynamically load TERASIC_PCIE_AVMM.DLL in C/C++ program. To load the DLL, please refer to the PCIe fundamental example below.
5. Call the SDK API to implement the desired application.

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

1.4 PCI Express Library API

Below shows the exported API in the TERASIC_PCIE_AVMM.DLL. The API prototype is defined in the TERASIC_PCIE_AVMM.h.
Note: the Linux library terasic_pcie_qsys.so also use the same API and header file.

■ PCIE_Open

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

Maximal read size is (4GB-1) bytes.

Prototype:

```
bool PCIE_DmaRead(
    PCIE_HANDLE hPCIE,
    PCIE_LOCAL_ADDRESS LocalAddress,
    void *pBuffer,
    uint32_t dwBufSize
    );
```

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.

dwBufSize:

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,
    uint32_t dwDataSize
    );
```

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.

dwDataSize:

Specify the byte number of data which will be written to FPGA.

Return Value:

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

PCIE_ConfigRead32

Function:

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

Prototype:

```
bool PCIE_ConfigRead32 (
```

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

1.5 PCIe Reference Design - Fundamental

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

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_Fundamental\demo_batch

The folder includes following files:

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

Demonstration Setup

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

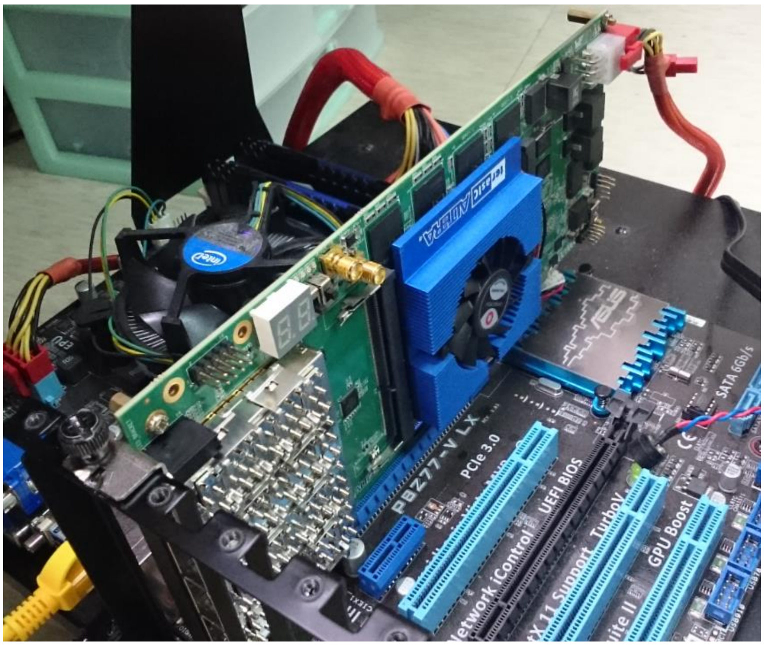


Figure 1-9 FPGA board installation on PC

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

   CDROM\Demonstration\PCIe_SW_KIT\Windows\PCIe_Driver.
4. Restart Windows
5. Make sure the Windows has detected the FPGA Board by checking the Windows Control panel as shown in **Figure 1-10**.

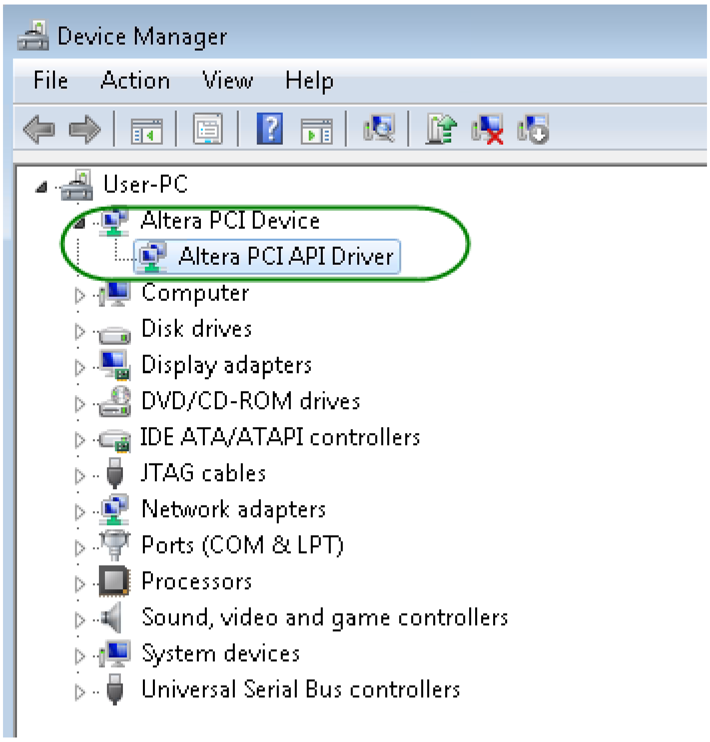


Figure 1-10 Screenshot for PCIe Driver

6. Goto windows_app folder, execute PCIE_FUNDAMENTAL.exe. A menu will appear as shown in **Figure 1-11**.

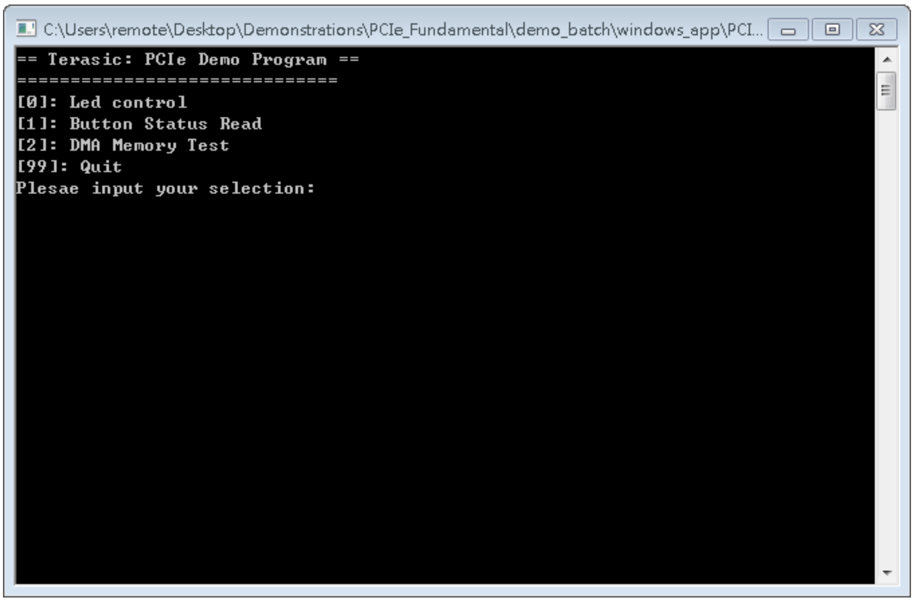


Figure 1-11 Screenshot of Program Menu

7. Type 0 followed by a ENTER key to select Led Control item, then input 3 (hex 0x03) will make all led on as shown in **Figure 1-12**. If input 0 (hex 0x00), all led will be turn off.

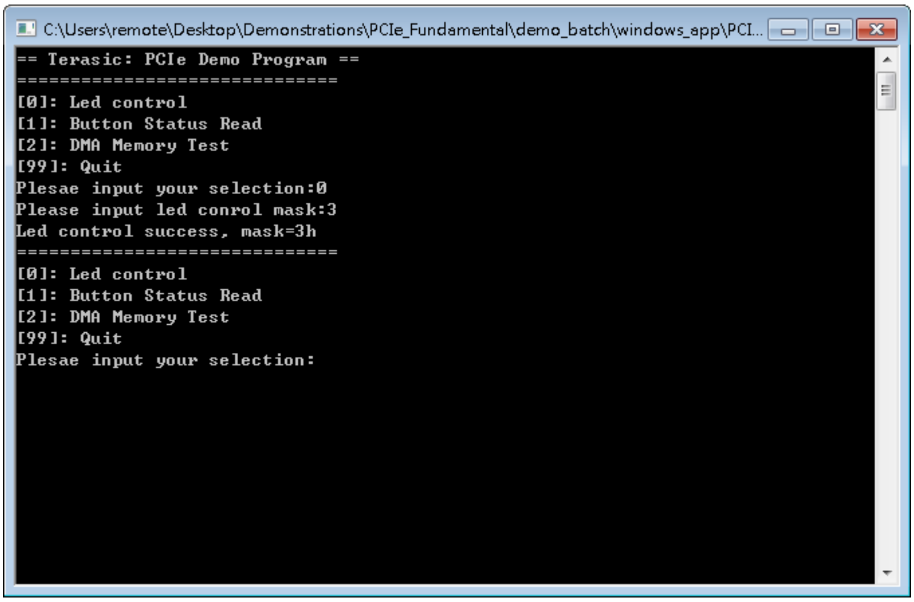


Figure 1-12 Screenshot of LED Control

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

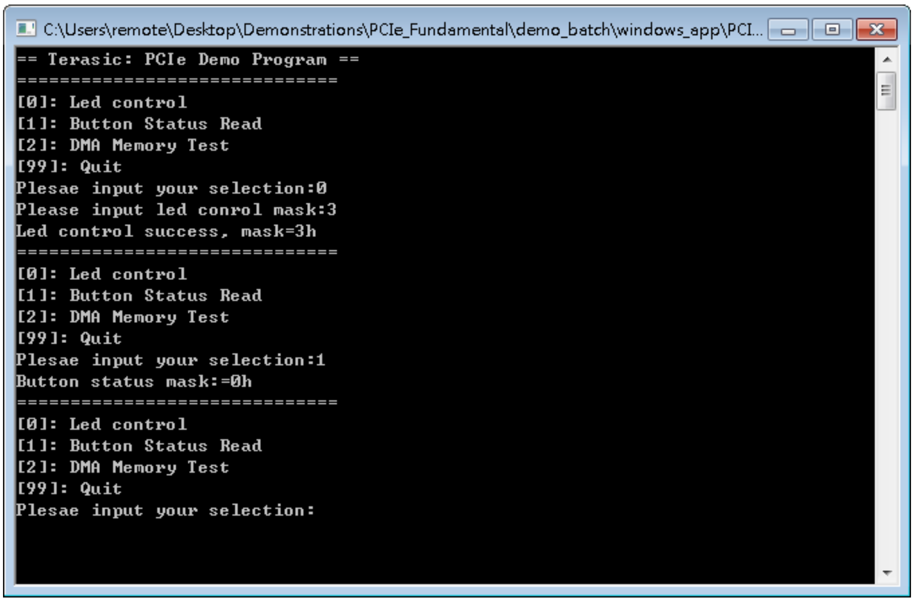


Figure 1-13 Screenshot of Button Status Report

9. Type 2 followed by an ENTER key to select DMA Testing item. The DMA test result will be report as shown in **Figure 1-14**.

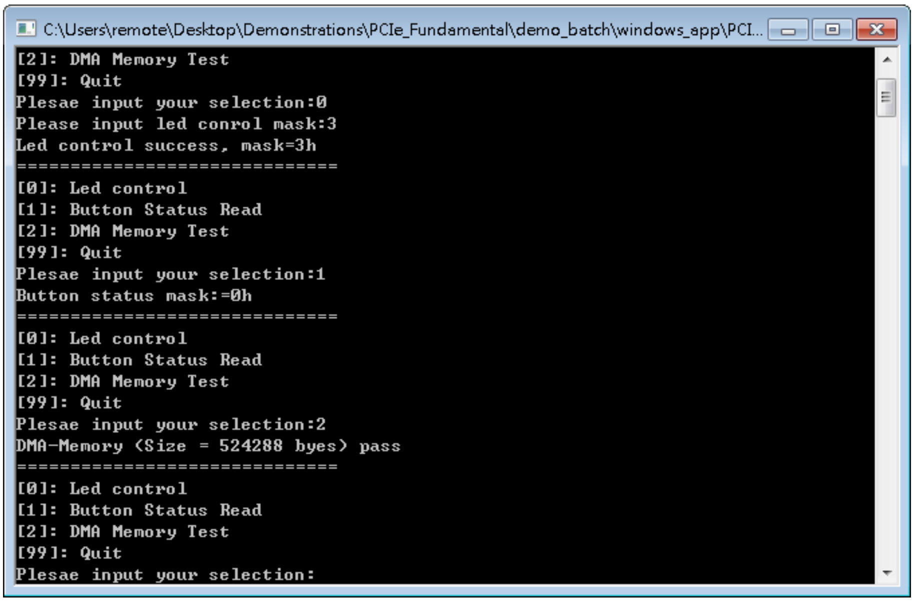


Figure 1-14 Screenshot of DMA Memory Test Result

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

■ Development Tools

- Quartus Prime 16.1.2 Standard Edition
- Visual C++ 2012

■ Demonstration Source Code Location

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

■ FPGA Application Design

Figure 1-15 shows the system block diagram in the FPGA system. In the Qsys, Altera 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.

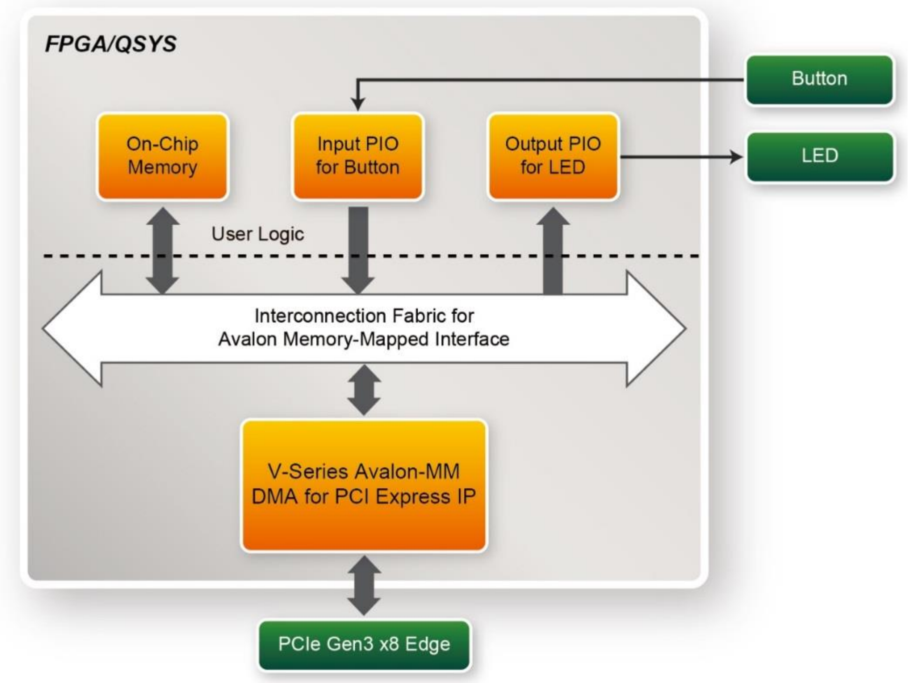


Figure 1-15 Hardware block diagram of the PCIe reference design

■ Windows Based Application Software Design

The application software project is built by Visual C++ 2012. The project includes the following major files:

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

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

```
#include "PCIE.h"

#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x4000010
#define DEMO_PCIE_IO_BUTTON_ADDR    0x4000020
#define DEMO_PCIE_MEM_ADDR          0x00000000

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

The base address of BUTTON and LED controllers are 0x4000010 and 0x4000020 based on PCIE_BAR4, in respectively. The on-chip memory base address is 0x00000000 relative to the DMA controller.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the TERASIC_PCIE_AVMM.DLL. Then, it call PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in PCIE_Open are defined in TERASIC_PCIE_AVMM.h. If developer change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value define in TERASIC_PCIE_AVMM.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:

```
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:

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

1.6 PCIe Reference Design - DDR3

The application reference design shows how to add DDR3 Memory Controllers for DDR3-A and DDR3-B into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 2GB data DMA for both DDR3 memory banks. Also, this demo shows how to call "PCIE_ConfigRead32" API to check PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_DDR3\demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_DDR3.sof
- Download Batch file: test.bat
- Windows Application Software folder : windows_app, includes
  - PCIE_DDR3.exe
  - TERASIC_PCIE_AVMM.DLL

■ Demonstration Setup

1. Install the FPGA board on your PC.
2. Configure FPGA with PCIe_DDR3.sof by executing the test.bat.
3. Install PCIe driver if necessary.
4. Restart Windows
5. Make sure the Windows has detected the FPGA Board by checking the Windows Control panel.
6. Goto windows_app folder, execute PCIE_DDR3.exe. A menu will appear as shown in **Figure 1-16**.

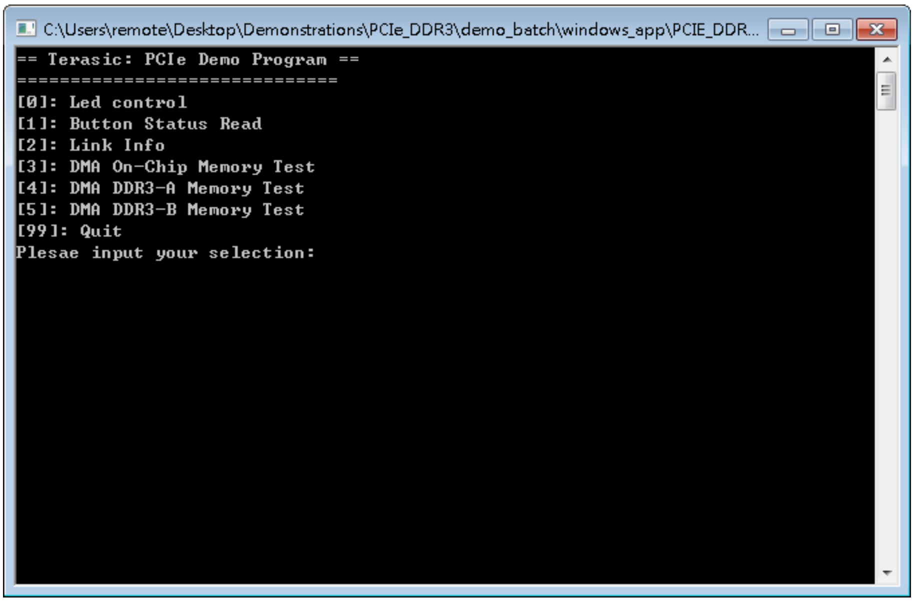


Figure 1-16 Screenshot of Program Menu

7. Type 2 followed by a ENTER key to select Link Info item. The PCIe link information will be shown as in **Figure 1-17**. Gen3 link speed and x4 link width are expected.

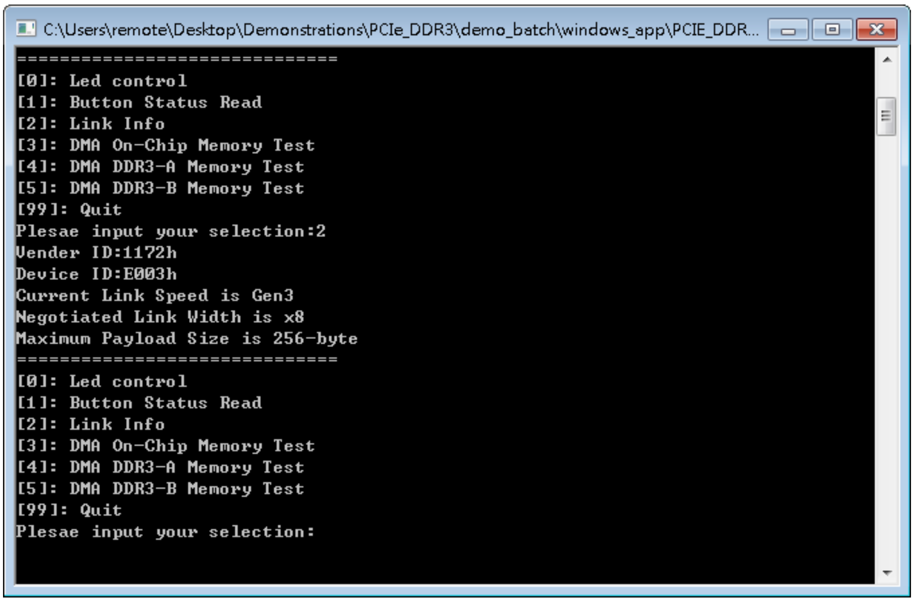
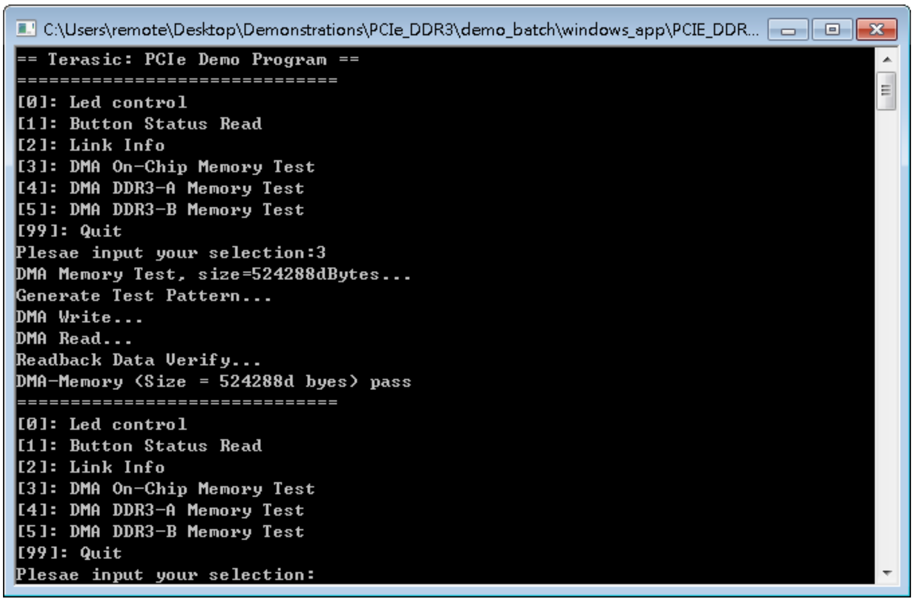


Figure 1-17 Screenshot of Link Info

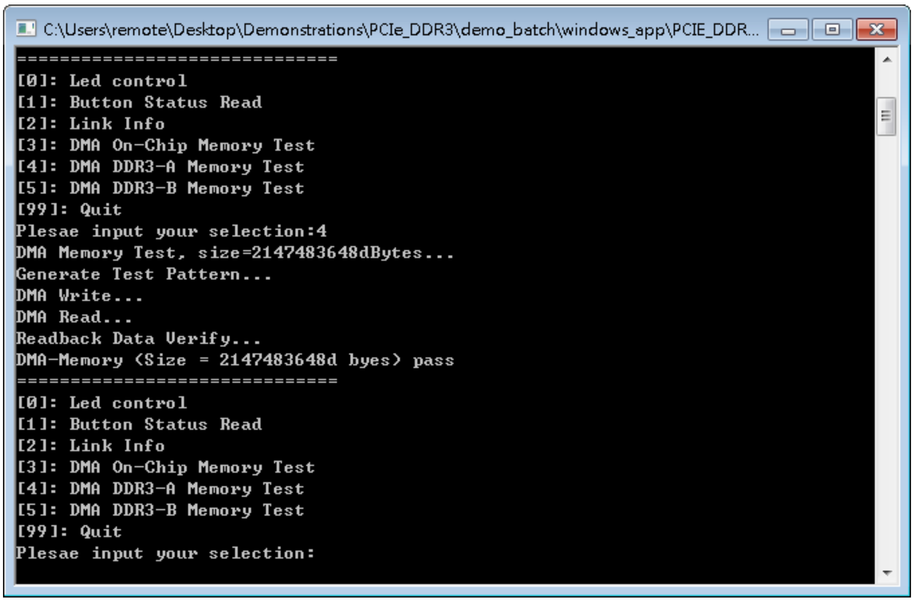
8. Type 3 followed by an ENTER key to select DMA On-Chip Memory Test item. The DMA write and read test result will be report as shown in **Figure 1-18**.



```
C:\Users\remote\Desktop\Demonstrations\PCIe_DDR3\demo_batch\windows_app\PCIE_DDR...
== Terasic: PCIe Demo Program ==
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:3
DMA Memory Test, size=524288dBytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory (Size = 524288d byes) pass
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 1-18 Screenshot of On-Chip Memory DMA Test Result

9. Type 4 followed by an ENTER key to select DMA DDR3-A Memory Test item. The DMA write and read test result will be report as shown in **Figure 1-19**.



```
C:\Users\remote\Desktop\Demonstrations\PCIe_DDR3\demo_batch\windows_app\PCIE_DDR...
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:4
DMA Memory Test, size=2147483648dBytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory (Size = 2147483648d byes) pass
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 1-19 Screenshot of DDR3-A Memory DAM Test Result

10. Type 5 followed by an ENTER key to select DMA DDR3-B Memory Test item. The DMA write and read test result will be report as shown in **Figure 1-20**.

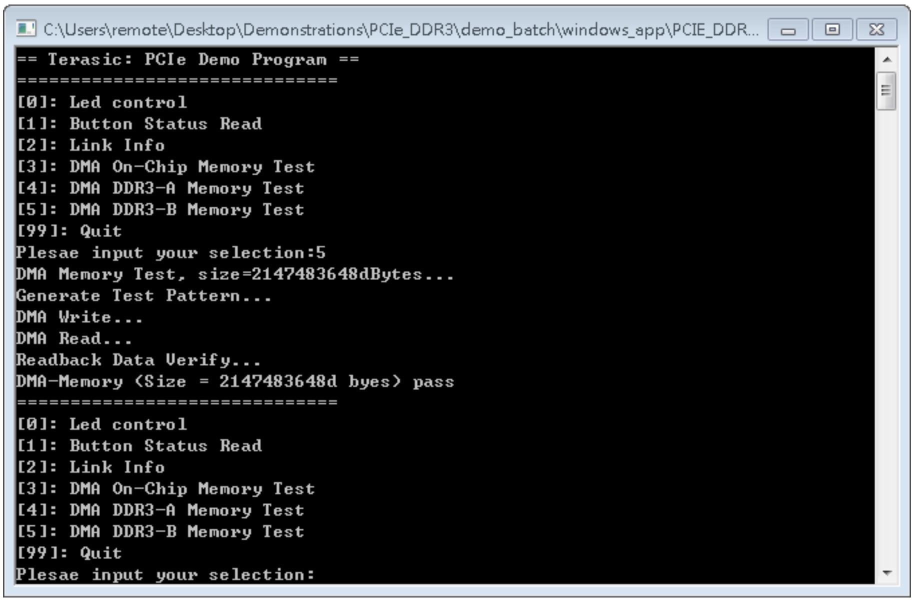


Figure 1-20 Screenshot of DDR3-B Memory DAM Test Result

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

■ Development Tools

- Quartus Prime 16.1.2 Standard Edition
- Visual C++ 2012

■ Demonstration Source Code Location

- Quartus Project: Demonstrations\PCIE_DDR3
- Visual C++ Project: Demonstrations\PCIe_SW_KIT\Windows\PCIe_DDR3

■ FPGA Application Design

Figure 1-21 shows the system block diagram in the FPGA system. In the Qsys, Altera 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.

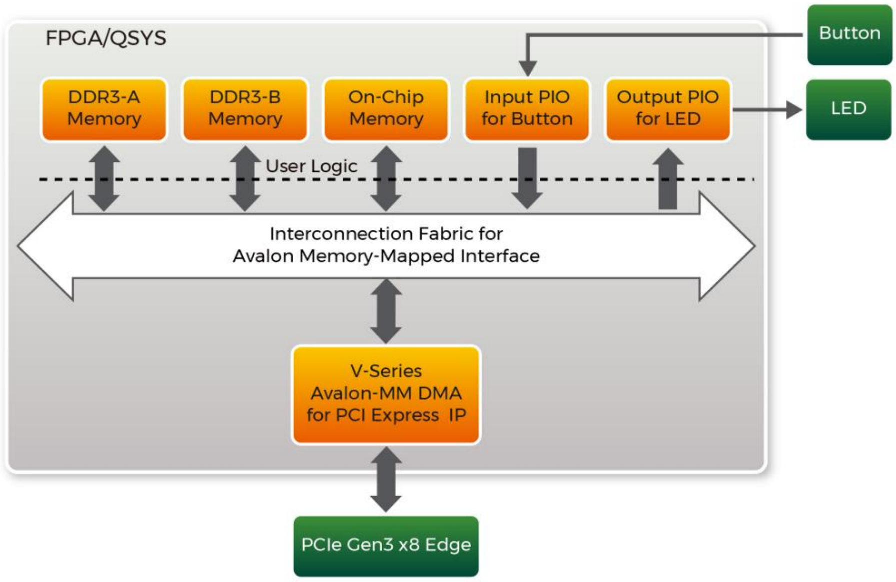


Figure 1-21 Hardware block diagram of the PCIe_DDR3 reference design

■ Windows Based Application Software Design

The application software project is built by Visual C++ 2012. The project includes the following major files:

Name	Description
PCIE_DDR3.cpp	Main program
PCIE.c	Implement dynamically load for TERAISC_PCIE_AVMM.DLL
PCIE.h	
TERASIC_PCIE_AVMM.h	SDK library file, defines constant and data structure

The main program PCIE_DDR3.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       0x4000010
#define DEMO_PCIE_IO_BUTTON_ADDR    0x4000020
#define DEMO_PCIE_ONCHIP_MEM_ADDR   0x00000000
#define DEMO_PCIE_DDR3A_MEM_ADDR    0x100000000
#define DEMO_PCIE_DDR3B_MEM_ADDR    0x200000000

#define ONCHIP_MEM_TEST_SIZE        (512*1024) //512KB
#define DDR3A_MEM_TEST_SIZE         (2*1024*1024*1024) //2GB
#define DDR3B_MEM_TEST_SIZE         (2*1024*1024*1024) //2GB
```

The base address of BUTTON and LED controllers are 0x4000010 and 0x4000020 based on PCIE_BAR4, in respectively. The on-chip memory base address is 0x00000000 relative to the DMA controller. The above definition is the same as those in PCIe Fundamental demo.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the TERASIC_PCIE_AVMM.DLL. Then, it call PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in PCIE_Open are defined in TERASIC_PCIE_AVMM.h. If developer change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value define in TERASIC_PCIE_AVMM.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:

```
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:

```
PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
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,  0x90, &Data32)){
  switch((Data32 >> 16) & 0x0F){
              case 1:
                  printf("Current Link Speed is Gen1\r\n");
                  break;
              case 2:
                  printf("Current Link Speed is Gen2\r\n");
                  break;
              case 3:
                  printf("Current Link Speed is Gen3\r\n");
                  break;
              default:
                  printf("Current Link Speed is Unknown\r\n");
                  break;
  }
  switch((Data32 >> 20) & 0x3F){
              case 1:
                  printf("Negotiated Link Width is x1\r\n");
                  break;
              case 2:
                  printf("Negotiated Link Width is x2\r\n");
                  break;
              case 4:
                  printf("Negotiated Link Width is x4\r\n");
                  break;
              case 8:
                  printf("Negotiated Link Width is x8\r\n");
                  break;
              case 16:
                  printf("Negotiated Link Width is x16\r\n");
                  break;
              default:
                  printf("Negotiated Link Width is Unknown\r\n");
                  break;
  }
}else{
  bPass = false;
}
```

Chapter 2

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 and FPGA communicate with each other through the PCI Express interface. V-Series Avalon-MM DMA for PCI Express IP is used in this demonstration. For detail about this IP, please refer to Intel FPGA document: ug_pcie_avmm_dma.pdf.

2.1 PCI Express System Infrastructure

Figure 2-1 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 V-Series Avalon-MM DMA for PCI Express. The application software on the PC side is developed by Terasic based on Altera's PCIe kernel mode driver.

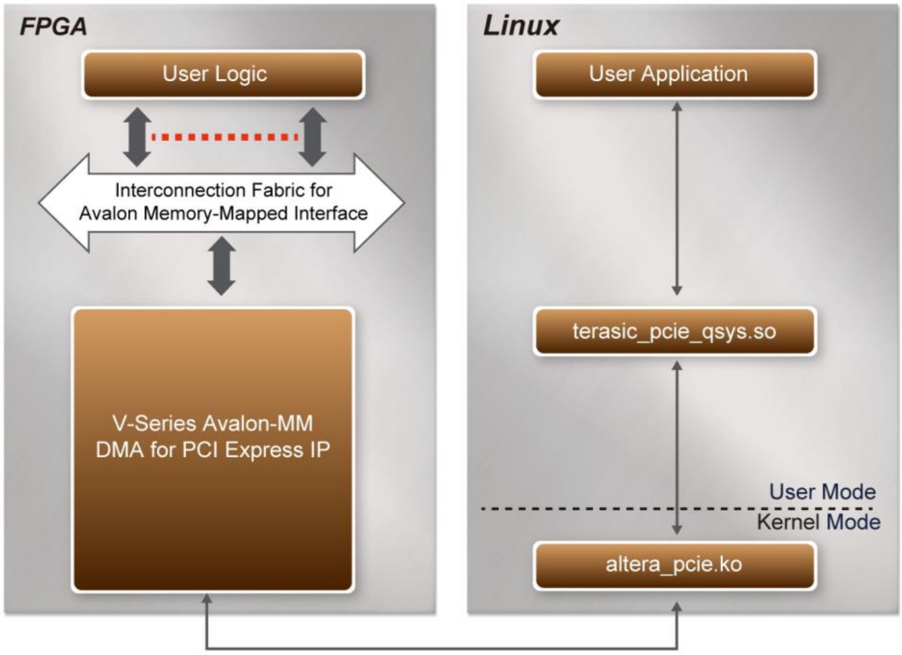


Figure 2-1 Infrastructure of PCI Express System

2.2 PC PCI Express Software SDK

The FPGA System CD contains a PC Windows based SDK to allow users to develop their 64-bit software application on 64-bits Linux. Ubuntu 16.04 is recommended. The SDK is located in the “CDROM/Demonstrations/PCIe_SW_KIT/Linux” folder which includes:

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

The kernel mode driver assumes the PCIe vendor ID (VID) is 0x1172 and the device ID (DID) is 0xE003. 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 project and rebuild the driver. The ID is defined in the file PCIe_SW_KIT/Linux/PCIe_Driver/altera_pcie_cmd.h.

The PCI Express Library is implemented as a single .so file named terasic_pcie_qsys.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, Altera AVMM DMA is required as the read and write operations are specified under the hardware design on the FPGA.

2.3 PCI Express Software Stack

Figure 2-2 shows the software stack for the PCI Express application software on 64-bit Linux. The PCIe library module terasic_pcie_qys.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 altera_pcie.ko kernel driver is provided by Altera.

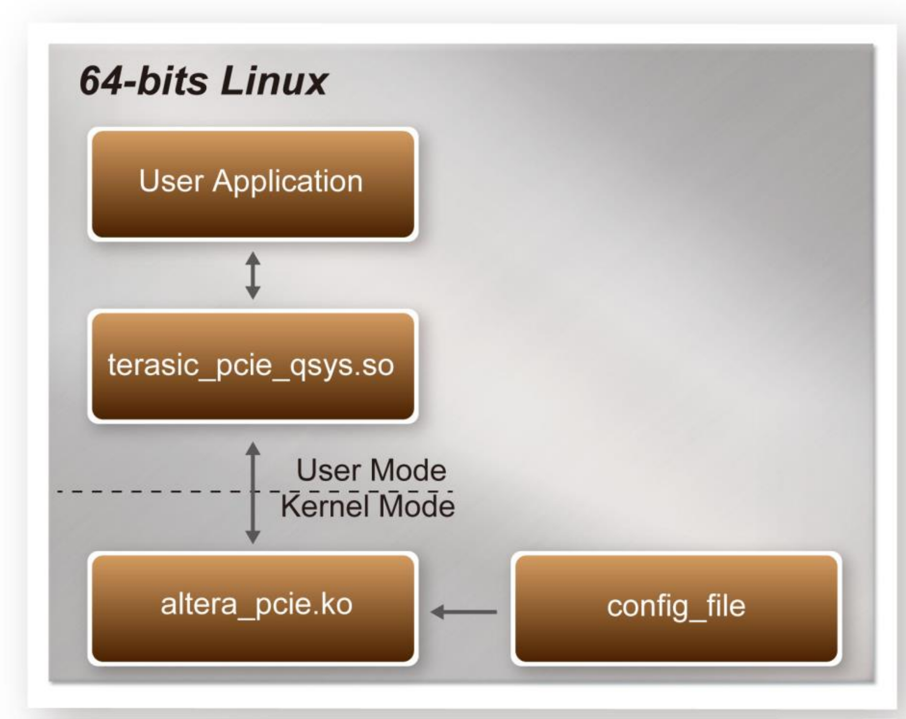


Figure 2-2 PCI Express Software Stack

■ Install PCI Express Driver on Linux

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

"CDROM/Demonstrations/PCIe_SW_KIT/Linux/PCIe_Driver"

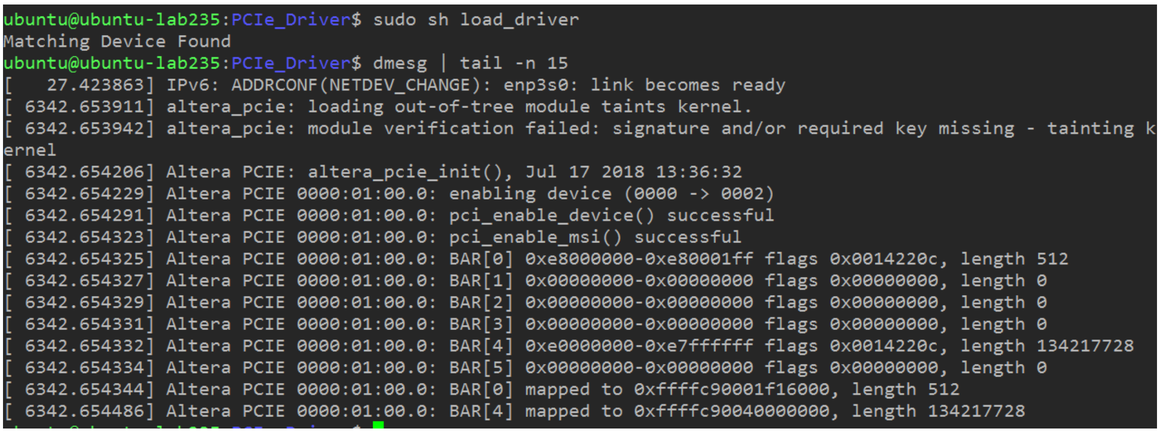
The folder includes the following files:

- altera_pcie.c
- altera_pcie.h
- altera_pcie_cmd.h
- Makefile
- load_driver
- unload
- config_file

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

1. Install the TR5-Lite on the PCIe slot of the host PC
2. Make sure Altera Programmer and USB-Blaster II driver are installed
3. Open a terminal and use "cd" command to goto the folder "CDROM/Demonstrations/PCIe_Fundamental/demo_batch".
4. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace “/home/ubuntu/intelFPGA/16.1/quartus” to your quartus installation path.

   ```
   export QUARTUS_ROOTDIR=/home/ubuntu/intelFPGA/16.1/quartus
   ```
5. Execute "sudo -E sh test.sh" command to configure the FPGA
6. Restart 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 altera_pcie.ko, and make sure driver is loaded successfully and FPGA is detected by the driver as shown in **Figure 2-3**.
   - make
   - sudo sh load_driver
   - dmesg | tail -n 15



```
ubuntu@ubuntu-lab235:PCIe_Driver$ sudo sh load_driver
Matching Device Found
ubuntu@ubuntu-lab235:PCIe_Driver$ dmesg | tail -n 15
[   27.423863] IPv6: ADDRCONF(NETDEV_CHANGE): enp3s0: link becomes ready
[ 6342.653911] altera_pcie: loading out-of-tree module taints kernel.
[ 6342.653942] altera_pcie: module verification failed: signature and/or required key missing - tainting k
ernel
[ 6342.654206] Altera PCIE: altera_pcie_init(), Jul 17 2018 13:36:32
[ 6342.654229] Altera PCIE 0000:01:00.0: enabling device (0000 -> 0002)
[ 6342.654291] Altera PCIE 0000:01:00.0: pci_enable_device() successful
[ 6342.654323] Altera PCIE 0000:01:00.0: pci_enable_msi() successful
[ 6342.654325] Altera PCIE 0000:01:00.0: BAR[0] 0xe8000000-0xe80001ff flags 0x0014220c, length 512
[ 6342.654327] Altera PCIE 0000:01:00.0: BAR[1] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 6342.654329] Altera PCIE 0000:01:00.0: BAR[2] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 6342.654331] Altera PCIE 0000:01:00.0: BAR[3] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 6342.654332] Altera PCIE 0000:01:00.0: BAR[4] 0xe0000000-0xe7ffffff flags 0x0014220c, length 134217728
[ 6342.654334] Altera PCIE 0000:01:00.0: BAR[5] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 6342.654344] Altera PCIE 0000:01:00.0: BAR[0] mapped to 0xffffc90001f16000, length 512
[ 6342.654486] Altera PCIE 0000:01:00.0: BAR[4] mapped to 0xffffc90040000000, length 134217728
```

Figure 2-3 Screenshot of install PCIe driver

■ Create a Software Application

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

- TERASIC_PCIE_AVMM.h
- terasic_pcie_qsys.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_AVMM.h in the C/C++ project.
3. Copy terasic_pcie_qsys.so to the folder where the project execution file is located.
4. Dynamically load terasic_pcie_qsys.so in C/C++ program. To load the terasic_pcie_qsys.so, please refer to the PCIe fundamental example below.
5. Call the library API to implement the desired application.

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

2.4 PCI Express Library API

The API is the same as Windows Library. Please refer to the section 1.4 PCI Express Library API in this document.

2.5 PCIe Reference Design – Fundamental

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

Demonstration Files Location

The demo file is located in the batch folder:

CDROM/Demonstrations/PCIe_Fundamental/demo_batch

The folder includes following files:

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

Demonstration Setup

1. Install the FPGA board on your PC as shown in **Figure 2-4**.

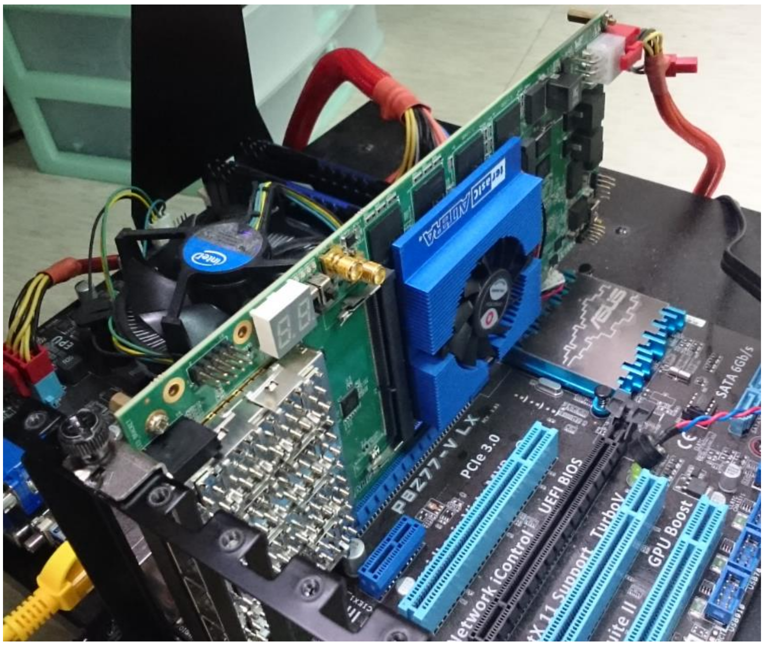


Figure 2-4 FPGA board installation on PC

2. Open a terminal and use "cd" command to goto "CDROM/Demonstrations/PCIe_Fundamental/demo_batch".
3. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set

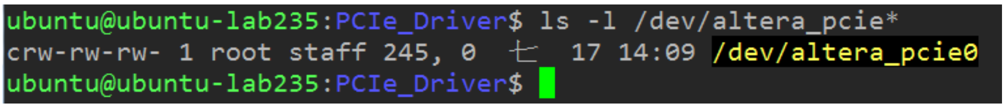
QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace /home/ubuntu/intelFPGA/16.1/quartus to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/ubuntu/intelFPGA/16.1/quartus
```

4. Execute "sudo -E sh test.sh" command to configure the FPGA
5. Restart Linux
6. Install PCIe driver. The driver is located in the folder:

   CDROM/Demonstration/PCIe_SW_KIT/Linux/PCIe_Driver.

7. Type “ls -l /dev/altera_pcie*” to make sure the Linux has detected the FPGA Board. If the FPGA board is detected, developers can find the /dev/altera_pcieX(where X is 0~255) in Linux file system as shown below.



8. Goto linux_app folder, execute PCIE_FUNDAMENTAL. A menu will appear as shown in **Figure 2-5**.

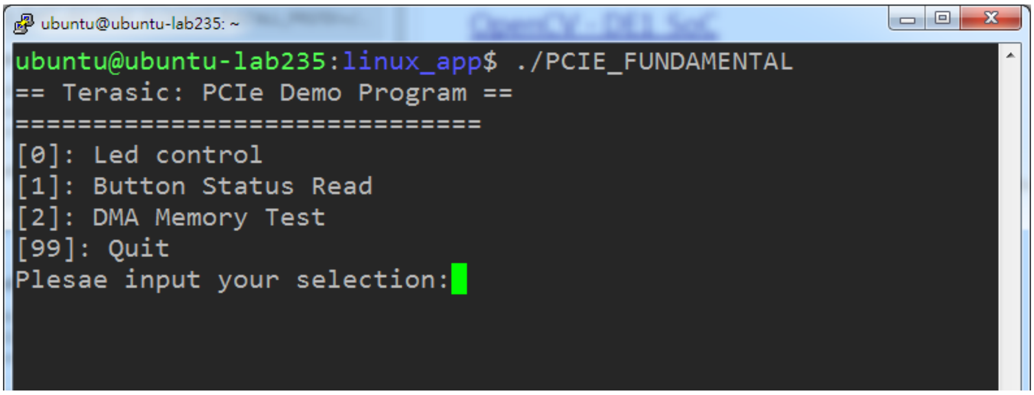


Figure 2-5 Screenshot of Program Menu

9. Type 0 followed by a ENTER key to select Led Control item, then input 15 (hex 0x0f) will make all led on as shown in **Figure 2-6**. If input 0 (hex 0x00), all led will be turn off.

```
ubuntu@ubuntu-lab235: ~
ubuntu@ubuntu-lab235:linux_app$ ./PCIE_FUNDAMENTAL
== Terasic: PCIe Demo Program ==
==============================
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:0
Please input led conrol mask:15
Led control success, mask=fh
==============================
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 2-6 Screenshot of LED Control

10. Type 1 followed by an ENTER key to select Button Status Read item. The button status will be report as shown in **Figure 2-7**.

```
ubuntu@ubuntu-lab235: ~
[99]: Quit
Plesae input your selection:0
Please input led conrol mask:15
Led control success, mask=fh
==============================
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:1
Button status mask:=0h
==============================
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 2-7 Screenshot of Button Status Report

11. Type 2 followed by an ENTER key to select DMA Testing item. The DMA test result will be report as shown in **Figure 2-8**.

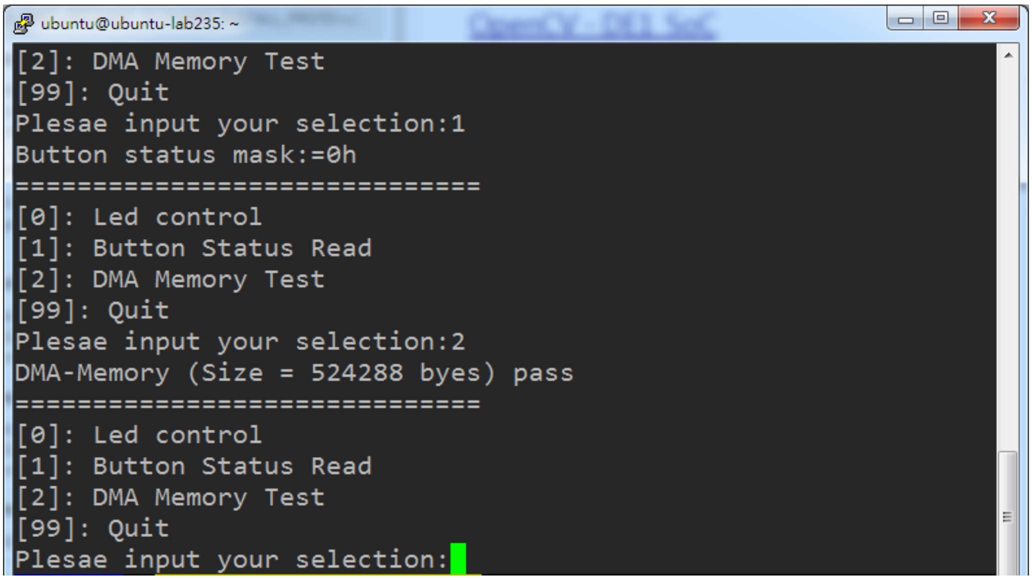


Figure 2-8 Screenshot of DMA Memory Test Result

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

■ Development Tools

- Quartus Prime 16.1.2 Standard Edition
- GNU Compiler Collection, Version 4.8 is recommend

■ Demonstration Source Code Location

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

■ FPGA Application Design

Figure 2-9 shows the system block diagram in the FPGA system. In the Qsys, Altera 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.

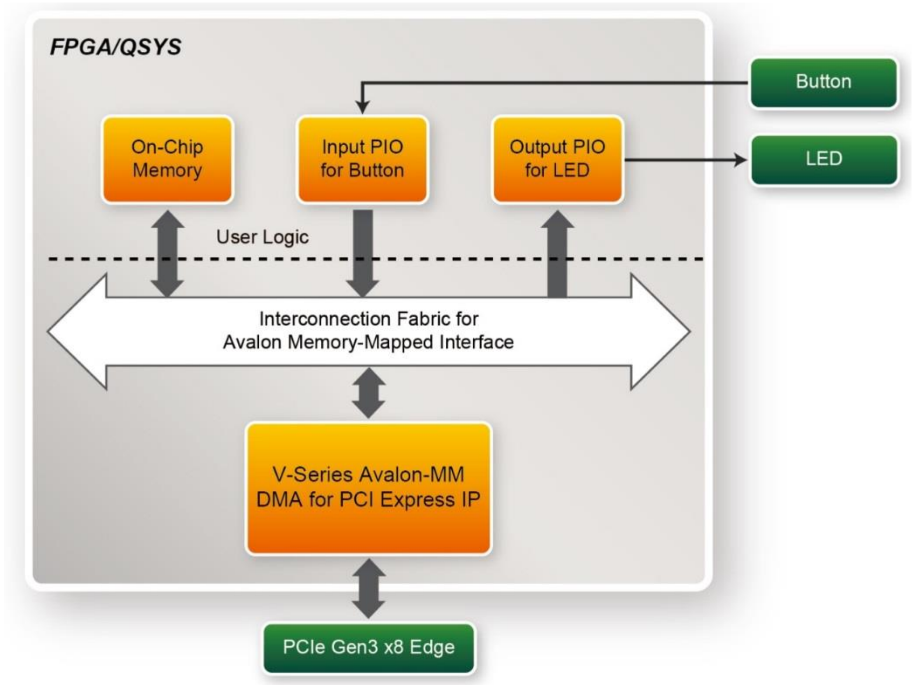


Figure 2-9 Hardware block diagram of the PCIe reference design

■ Linux Based Application Software Design

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

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

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

```
#include "PCIE.h"

#define DEMO_PCIE_USER_BAR          PCIE_BAR4
#define DEMO_PCIE_IO_LED_ADDR       0x4000010
#define DEMO_PCIE_IO_BUTTON_ADDR    0x4000020
#define DEMO_PCIE_MEM_ADDR          0x00000000

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

The base address of BUTTON and LED controllers are 0x4000010 and 0x4000020 based on PCIE_BAR4, in respectively. The on-chip memory base address is 0x00000000 relative to the DMA controller.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the terasic_pcie_qsys.so. Then, it call PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in PCIE_Open are defined in TERASIC_PCIE_AVMM.h. If developer change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value define in TERASIC_PCIE_AVMM.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:

```
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:

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

2.6 PCIe Reference Design - DDR3

The application reference design shows how to add DDR3 Memory Controllers for DDR3-A and DDR3-B into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 2GB data DMA for both DDR3 memory banks. Also, this demo shows how to call "PCIE_ConfigRead32" API to check PCIe link status.

Demonstration Files Location

The demo file is located in the batch folder:

CDROM/Demonstrations/PCIe_DDR3/demo_batch

The folder includes following files:

- FPGA Configuration File: PCIe_DDR3.sof
- Download Batch file: test.sh
- Linux Application Software folder : linux_app, includes
  - PCIE_DDR3
  - terasic_pcie_qsys.so

Demonstration Setup

1. Install the FPGA board on your PC.
2. Open a terminal and use "cd" command to goto "CDROM/Demonstrations/PCIe_DDR3/demo_batch".
3. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace /home/ubuntu/intelFPGA/16.1/quartus to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/ubuntu/intelFPGA/16.1/quartus
```

4. Execute "sudo -E sh test.sh" command to configure the FPGA
5. Restart Linux
6. Install PCIe driver.
7. Make sure the Linux has detected the FPGA Board.
8. Goto linux_app folder, execute PCIE_DDR3. A menu will appear as shown in **Figure 2-10**.

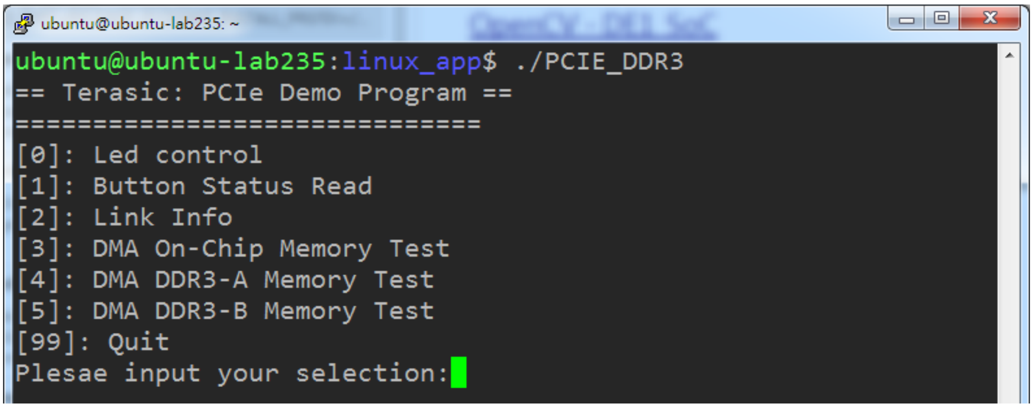


Figure 2-10 Screenshot of Program Menu

9. Type 2 followed by an ENTER key to select Link Info item. The PCIe link information will be shown as in **Figure 2-11**. Gen3 link speed and x8 link width are expected.

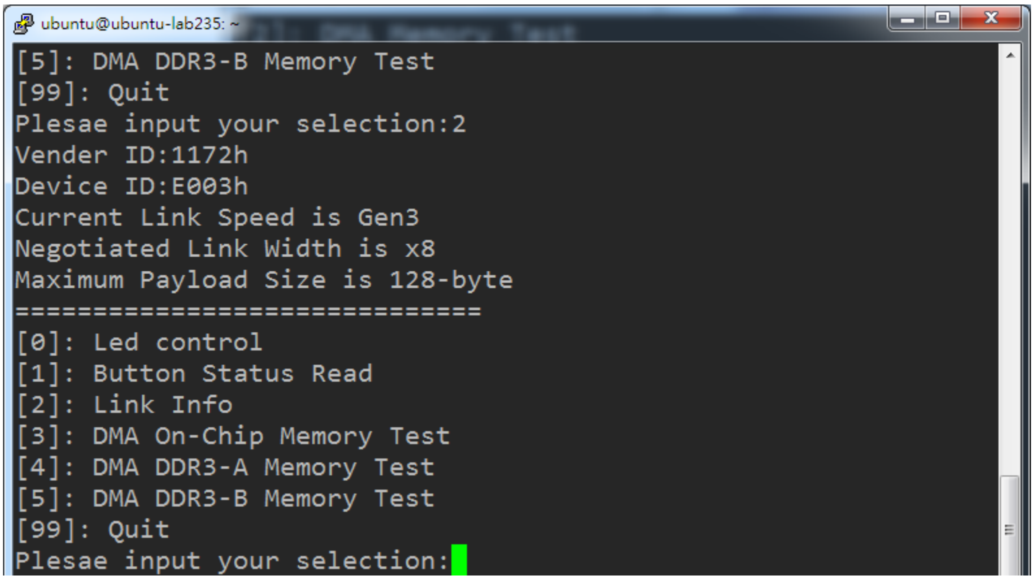


Figure 2-11 Screenshot of Link Info

10. Type 3 followed by an ENTER key to select DMA On-Chip Memory Test item. The DMA write and read test result will be report as shown in **Figure 2-12**.

```
ubuntu@ubuntu-lab235: ~
[99]: Quit
Plesae input your selection:3
DMA Memory Test, Address = 0x0, Size = 0x80000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x0, Size = 0x80000 bytes pass
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 2-12 Screenshot of On-Chip Memory DMA Test Result

11. Type 4 followed by an ENTER key to select DMA DDR3-A Memory Test item. The DMA write and read test result will be report as shown in **Figure 2-13**.

```
ubuntu@ubuntu-lab235: ~
[99]: Quit
Plesae input your selection:4
DMA Memory Test, Address = 0x100000000, Size = 0x80000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x100000000, Size = 0x80000000 bytes pass
==============================
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR3-A Memory Test
[5]: DMA DDR3-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 2-13 Screenshot of DDR3-A Memory DAM Test Result

12. Type 5 followed by an ENTER key to select DMA DDR3-B Memory Test item. The DMA write and read test result will be report as shown in **Figure 2-14**.

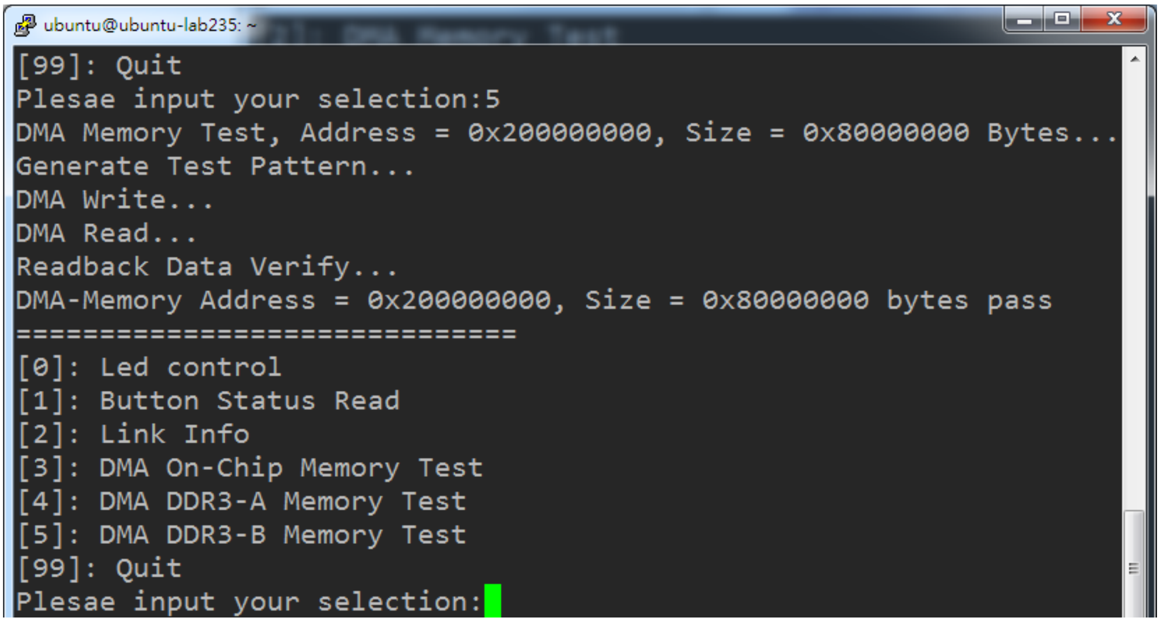


Figure 2-14 Screenshot of DDR3-B Memory DAM Test Result

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

■ Development Tools

- Quartus Prime 16.1.2 Standard Edition
- GNU Compiler Collection, Version 4.8 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstrations/PCIE_DDR3
- C++ Project: Demonstrations/PCIe_SW_KIT/Linux/PCIe_DDR3

■ FPGA Application Design

Figure 2-15 shows the system block diagram in the FPGA system. In the Qsys, Altera 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.

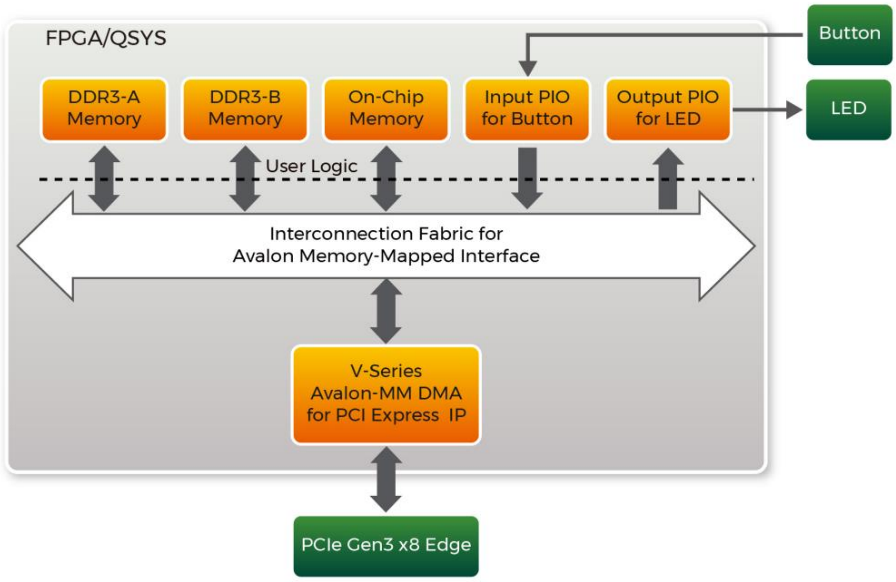


Figure 2-15 Hardware block diagram of the PCIe_DDR3 reference design

■ Linux Based Application Software Design

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

Name	Description
PCIE_DDR3.cpp	Main program
PCIE.c	Implement dynamically load for terasic_pcie_qsys.so library file
PCIE.h	
TERASIC_PCIE_AVMM.h	SDK library file, defines constant and data structure

The main program PCIE_DDR3.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       0x4000010
#define DEMO_PCIE_IO_BUTTON_ADDR    0x4000020
#define DEMO_PCIE_ONCHIP_MEM_ADDR   0x00000000
#define DEMO_PCIE_DDR3A_MEM_ADDR    0x100000000
#define DEMO_PCIE_DDR3B_MEM_ADDR    0x200000000

#define ONCHIP_MEM_TEST_SIZE        (512*1024) //512KB
#define DDR3A_MEM_TEST_SIZE         (2*1024*1024*1024) //2GB
#define DDR3B_MEM_TEST_SIZE         (2*1024*1024*1024) //2GB
```

The base address of BUTTON and LED controllers are 0x4000010 and 0x4000020 based on PCIE_BAR4, in respectively. The on-chip memory base address is 0x00000000 relative to the DMA controller. The above definition is the same as those in PCIe Fundamental demo.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the terasic_pcie_qsys.so. Then, it call PCIE_Open to open the PCI Express driver. The constant DEFAULT_PCIE_VID and DEFAULT_PCIE_DID used in PCIE_Open are defined in TERASIC_PCIE_AVMM.h. If developer change the Vendor ID and Device ID and PCI Express IP, they also need to change the ID value define in TERASIC_PCIE_AVMM.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:

```
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:

```
PCIE_DmaWrite(hPCIe, LocalAddr, pWrite, nTestSize);
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,  0x90, &Data32)){
  switch((Data32 >> 16) & 0x0F){
              case 1:
                  printf("Current Link Speed is Gen1\r\n");
                  break;
              case 2:
                  printf("Current Link Speed is Gen2\r\n");
                  break;
              case 3:
                  printf("Current Link Speed is Gen3\r\n");
                  break;
              default:
                  printf("Current Link Speed is Unknown\r\n");
                  break;
  }
  switch((Data32 >> 20) & 0x3F){
              case 1:
                  printf("Negotiated Link Width is x1\r\n");
                  break;
              case 2:
                  printf("Negotiated Link Width is x2\r\n");
                  break;
              case 4:
                  printf("Negotiated Link Width is x4\r\n");
                  break;
              case 8:
                  printf("Negotiated Link Width is x8\r\n");
                  break;
              case 16:
                  printf("Negotiated Link Width is x16\r\n");
                  break;
              default:
                  printf("Negotiated Link Width is Unknown\r\n");
                  break;
  }
}else{
  bPass = false;
}
```