FLIK

RTL Development Guide

terasIC
www.terasic.com

Contents

Chapter 1 *Overview*.....4

1.1 System Requirement.....4

1.2 CD Content.....4

1.3 Example Code.....4

1.4 EPCQ Programming for Windows.....5

Chapter 2 *Getting Started*.....6

2.1 Quartus Prime Installation.....6

2.2 JTAG Chain Test.....6

2.3 PCIe Test.....7

2.4 Power Monitor Utility (Windows).....7

Chapter 3 *PCI Express Reference for Windows*.....9

3.1 PCI Express System Infrastructure.....9

3.2 PCI Express Software SDK.....10

3.3 PCI Express Software Stack.....10

3.4 PCI Express Library API.....16

3.5 PCIe Reference Design - DDR4.....21

Chapter 4 *PCI Express Reference Design for Linux*.....28

4.1 PCI Express System Infrastructure.....28

4.2 PCI Express Software SDK 29

4.3 PCI Express Software Stack 29

4.4 PCI Express Library API 32

4.5 PCIe Reference Design - DDR4 32

Chapter 1

Overview

This document describes how to setup a development environment for RTL developer and explain the PCIe Example design includes in this Kit.

1.1 System Requirement

The following items are required for the development of FPGA RTL Design:

- Host PC with Thunderbolt 3 Port
- Quartus Prime Pro 19.2 or later Installed
- USB Blaster II driver installed

1.2 CD Content

Table 1-1 shows the CD content of the RTL Design Software Package.

Table 1-1 CD Content

Example Folder	Description
Demonstration	PCIe Example Code and Host Driver
Manual	This manual
Datasheet	Datasheet of major components on FLIK mainboard
Schematic	FLIK mainboard schematic

1.3 Example Code

FLIK Kit CD include a PCIe Example Project. The project contains:

- Quartus PCIe Project Source Code
- Application Source Code for Windows and Linux.
- PCIe Driver binary Code for Linux and Windows
- EPCQ Programming Batch for Windows

The Quartus PCIe example located in the folder:

RTL_Package/demonstration/PCIe_DDR4. The Host Application source code and PCIe driver are located in the folder RTL_Package/demonstration/PCIe_SW_KIT.
The EPCQ Programming Batch is located in the folder:
RTL_Package/demonstration/PCIe_DDR4/program_mt25q.

1.4 EPCQ Programming for Windows

The procedures to programming EPCQ is:

1. Go to folder RTL_Package/demonstration/PCIe_DDR4/program_mt25q
2. Overwrite the default_code.sof with your .sof.
3. Run “test.bat”. A console manual appears as shown in **Figure 1-1**.
4. Enter “2” to select “convert .sof to .jic”
5. Enter “3” to select “programming .jic to MT25QU01G”

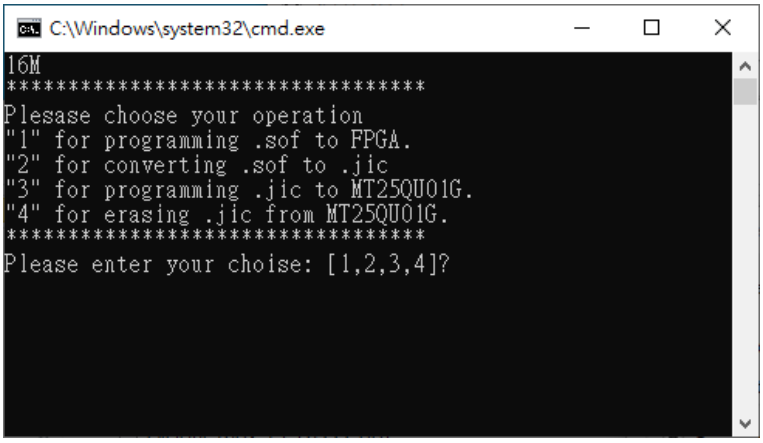


Figure 1-1 EPCQ Programming Menu

Chapter 2

Getting Started

Before developers starting RTL design and debug, development software should be installed first. Quartus Prime is required to compile user's RTL design. USB Blaster II driver is required for FPGA configuration and debugging by Signal Tap.

2.1 Quartus Prime Installation

Quartus Prime Pro 19.2 or later software can be download from:
https://fpgasoftware.intel.com/?edition=pro

For USB Blaster II driver installation guide on Windows, please see:
http://www.terasic.com.tw/wiki/Altera_USB_Blaster_II_Driver_Installation_Instructions

For USB Blaster II driver installation guide on Linux, please see:
https://rocketboards.org/foswiki/Documentation/UsingUSBBlasterUnderLinux

2.2 JTAG Chain Test

Before perform this test, please make sure the thunderbolt 3 driver is installed well on your Host PC. For details for installing thunderbolt 3 driver, please refer to the Getting Started Guide include in this Kit.

Quartus Prime programmer can be used performance JTAG Chain test to make sure thunderbolt 3 driver, USB-Blaster II driver, and FPGA JTAG work well. Here is the procedure to test JTAG Chain:

1. Plug 12V DC to FLIK to power on FLIK.
2. Connect FLIK to Host PC Thunderbolt 3 port by a Thunderbolt 3 cable.
3. Launch Quartus Prime Programmer

4. In Quartus Prime Programmer GUI, click “Hardware Setup” to find USB Blaster II
5. In Quartus Prime Programmer GUI, click “Auto Detect” to find Arria 10 FPGA

2.3 PCIe Test

Before performance this test, please make the thunderbolt 3 driver is installed well on your Host PC. Here is the major procedure to perform PCIe test:

1. Plug 12V DC to FLIK to power on FLIK.
2. Connect FLIK to Host PC Thunderbolt 3 port by a Thunderbolt 3 cable.
3. Configure FPGA as a PCIe Endpoint.
4. Remove and Plug the thunderbolt 3 cable
5. Install PCIe Driver
6. Launch the PCIe Test Software and Test PCIe.

For Windows Host, please install PCIe driver by refer to **Chapter 3.3** . For PCIe function test, please refer to **Chapter 3.5** . For Linux Host, please install PCIe driver by refer to **Chapter 4.3** . For PCIe function test, please refer to **Chapter 4.5** .

2.4 Power Monitor Utility (Windows)

The FLIK kit includes a Power Monitor Software Utility which can monitor the power and temperature status of the FLIK System. The software only available for Windows, the software is available on http://flik.terasic.com/cd. Here is the procedure to launch Power Monitor:

1. Plug 12V DC to FLIK to power on FLIK.
2. Connect FLIK to Host PC Thunderbolt 3 port by a Thunderbolt 3 cable.
3. Launch Power Monitor.
4. Power Monitor appears as shown in **Figure 2-1**.

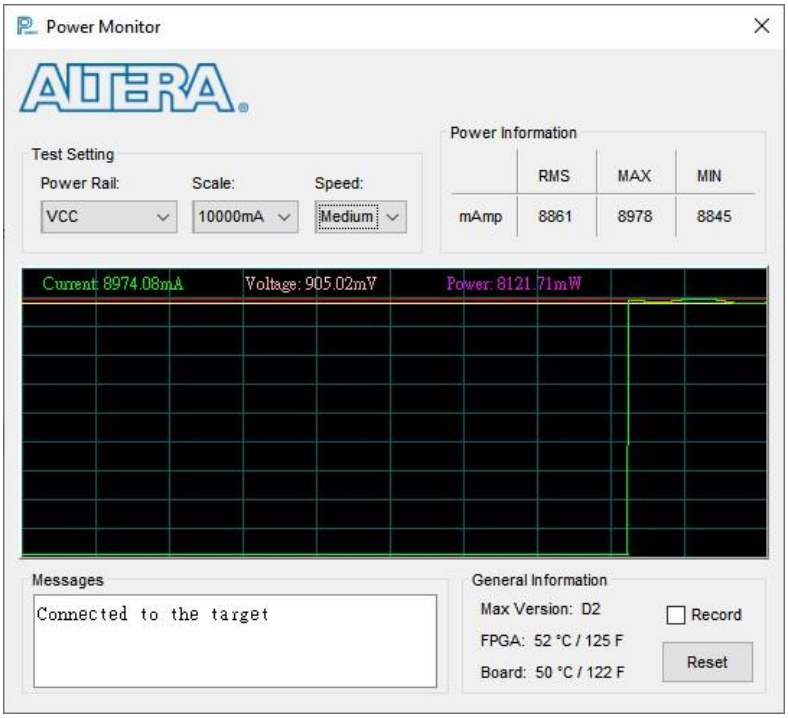


Figure 2-1 Power Monitor Screenshot

Chapter 3

PCI Express Reference for Windows

This document will show how the Windows PC and FPGA communicate with each other through the PCI Express interface. Arria 10 Hard IP for PCI Express with Avalon-MM DMA IP is used in this demonstration. For detail about this IP, please refer to Intel document ug_a10_pcie_avmm_dma.pdf.

3.1 PCI Express System Infrastructure

Figure 3-2 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 Arria 10 Hard IP for PCI Express with Avalon-MM DMA. The application software on the PC side is developed by Terasic based on Altera's PCIe kernel mode driver.

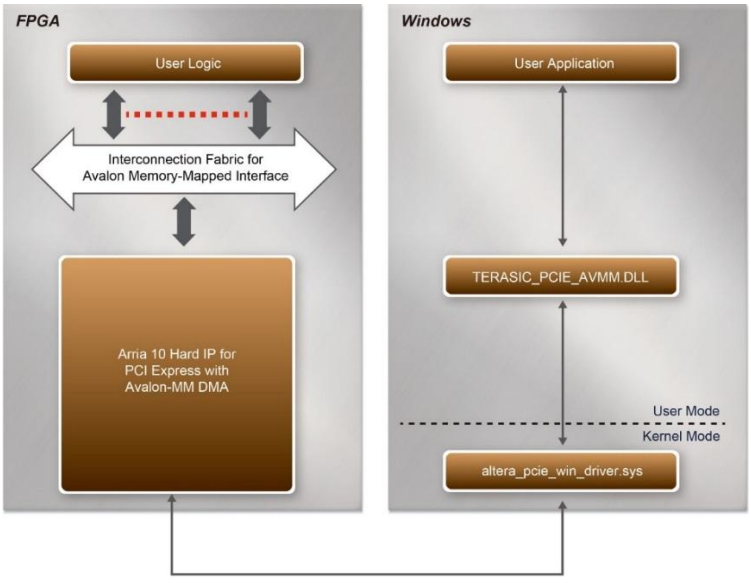


Figure 3-2 Infrastructure of PCI Express System

3.2 PCI Express Software SDK

The FPGA System CD contains a PC Windows based SDK to allow users to develop their 64-bit software application on 64-bits Windows 10. The SDK is located in the "SystemCD\demonstration\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 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 INF file accordingly.

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

3.3 PCI Express Software Stack

Figure 3-3 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 Intel.

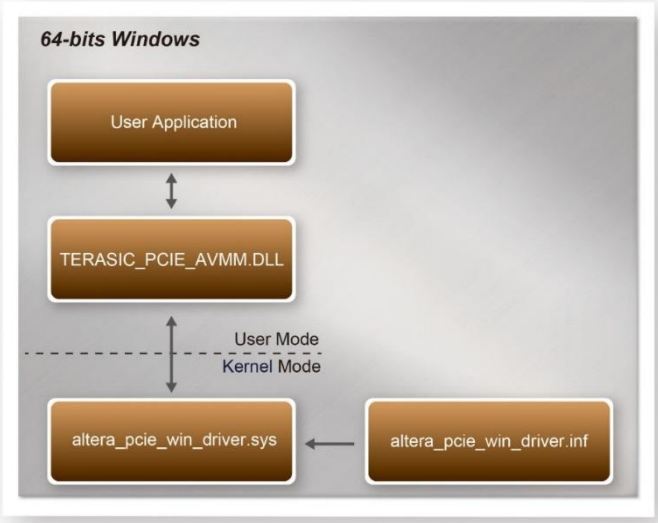


Figure 3-3 PCI Express Software Stack

■ Install PCI Express Driver on Windows

The PCIe driver is locate in the folder:

"SystemCD\demonstration\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. Make sure the Thunderbolt 3 driver is installed well on the Host PC.
2. Make sure Quartus Prime Programmer and USB-Blaster II driver are installed.
3. Connect 12V DC to FLIK to power on FLIK.
4. Connect the FLIK to the Host PC via a Thunderbolt 3 Cable.
5. Execute test.bat in "SystemCD\demonstration\PCIe_DDR4l\demo_batch" to configure the FPGA
6. Remove and plug the Thunderbot3 cable.
7. 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 3-4** Move the mouse cursor to the PCI Device item and right click it to select the Update Driver Software... item.

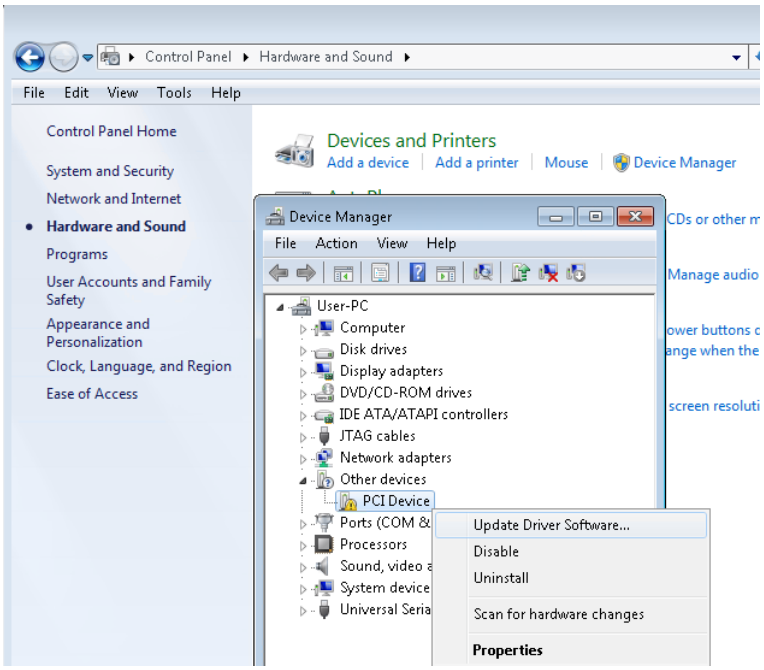


Figure 3-4 Screenshot of launching Update Driver Software… dialog

8. In the **How do you want to search for driver software** dialog, click **Browse my computer for driver software** item, as shown in **Figure 3-5**.

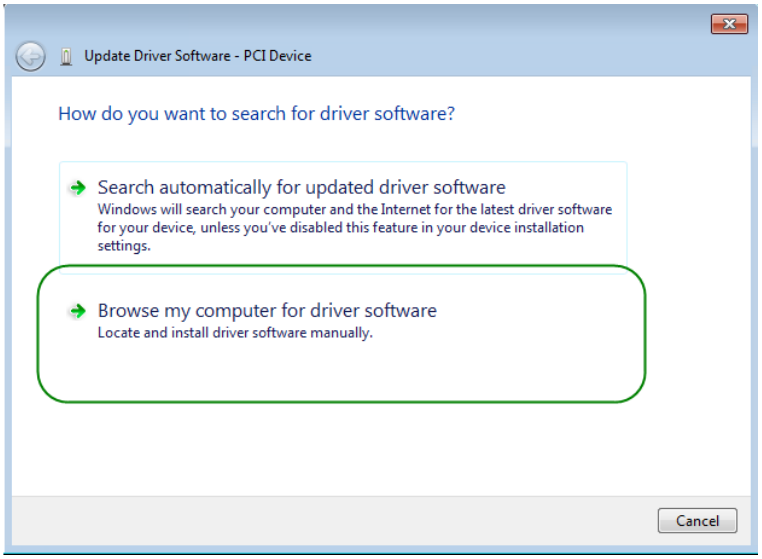


Figure 3-5 Dialog of Browse my computer for driver software

9. 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 3-6**. Click the **Next** button.

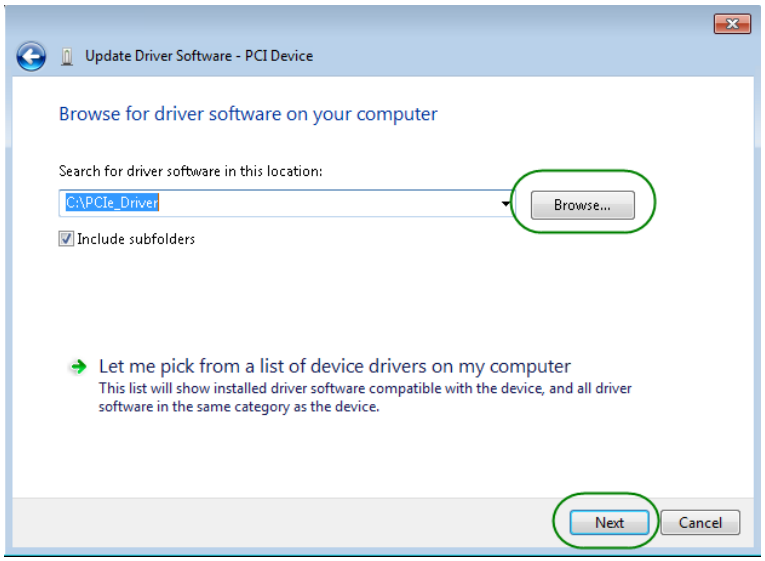
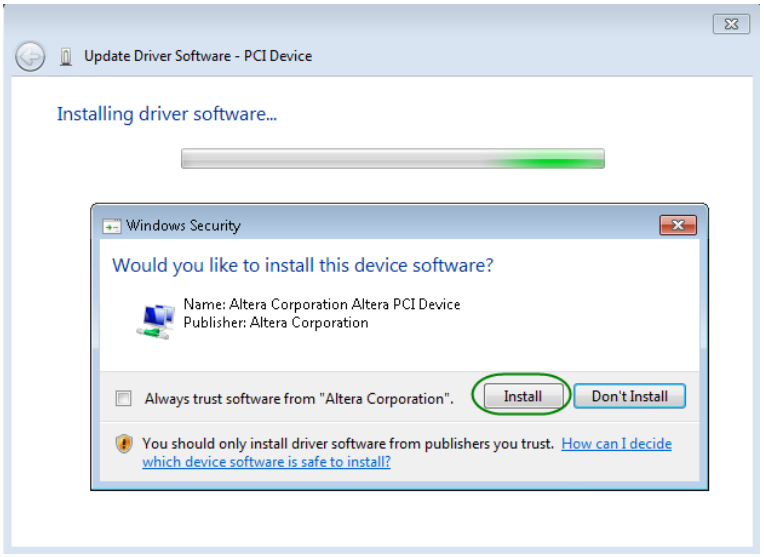


Figure 3-6 Browse for driver software on your computer

10. When the **Windows Security** dialog appears, as shown



11. **Figure 3-7**, click the **Install** button.

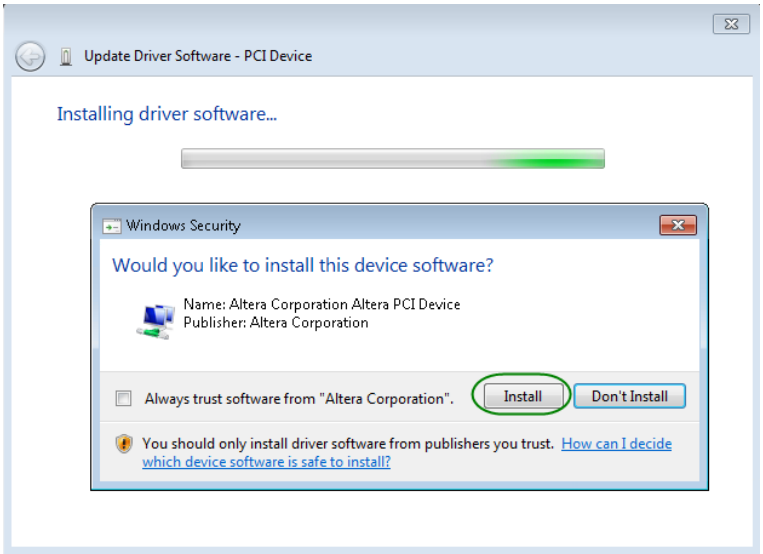
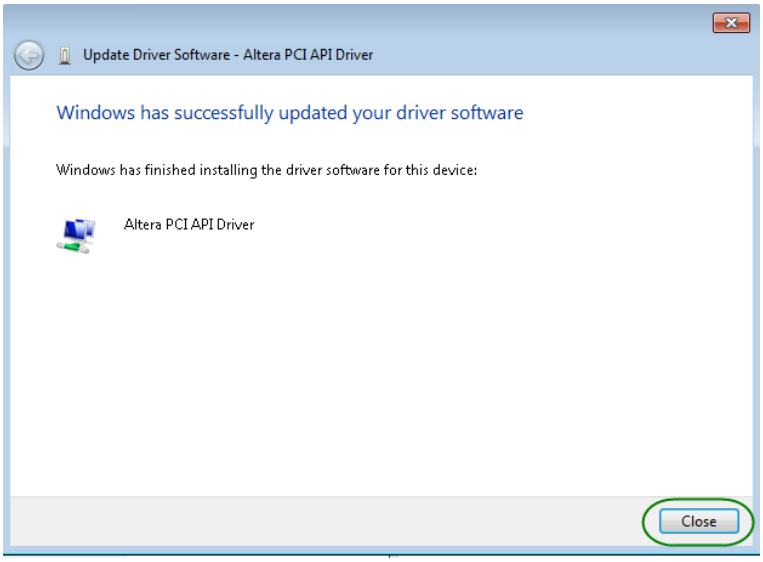


Figure 3-7 Click Install in the dialog of Windows Security

12. When the driver is installed successfully, the successfully dialog will appear, as shown in **Figure 3-8**. Click the **Close** button.



13.

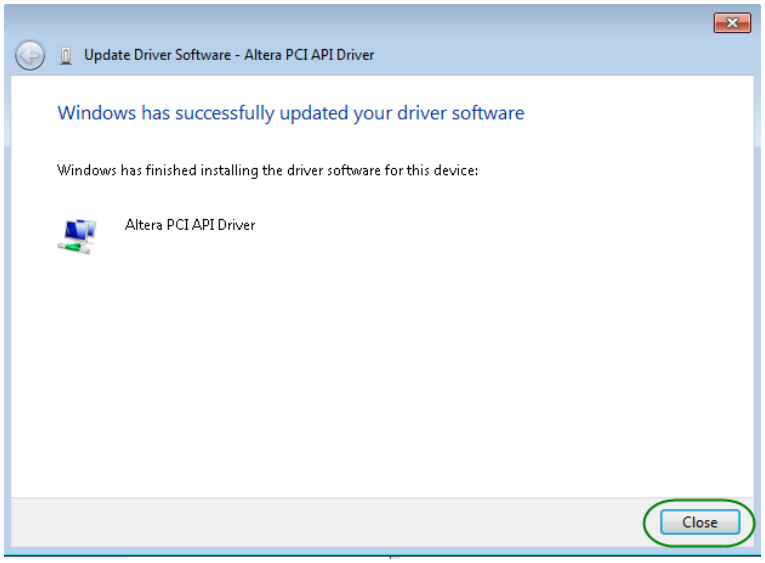


Figure 3-8 Click Close when the installation of Altera PCI API Driver is complete

14. Once the driver is successfully installed, users can see the **Altera PCI API Driver** under the device manager window, as shown in **Figure 3-9**.

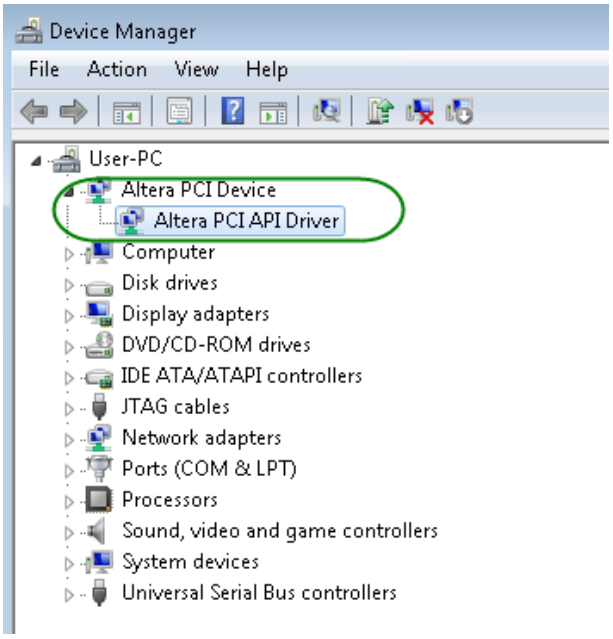


Figure 3-9 Altera PCI API Driver in Device Manager

■ Create a Software Application

All the files needed to create a PCIe software application are located in the directory

SystemCD\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_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_AVMM.dll API. The details of API are described below:

3.4 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(
    Uint16_t wVendorID,
    Uint16_t wDeviceID,
    Uint16_t wCardNum);
```

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.

wCardNum:

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 hFPGA);
```

Parameters:

hFPGA:

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 hFPGA,
    PCIE_BAR PciBar,
    PCIE_ADDRESS PciAddress,
    uint32_t *pdwData);
```

Parameters:

hFPGA:

A PCIe handle return by PCIE_Open function.

PciBar:

Specify the target BAR.

PciAddress:

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 hFPGA,
    PCIE_BAR PciBar,
    PCIE_ADDRESS PciAddress,
    uint32_t dwData);
```

Parameters:

hFPGA:

A PCIe handle return by PCIE_Open function.

PciBar:

Specify the target BAR.

PciAddress:

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 hFPGA,
    PCIE_BAR PciBar,
    PCIE_ADDRESS PciAddress,
    uint8_t *pByte);
```

Parameters:

hFPGA:

A PCIe handle return by PCIE_Open function.

PciBar:

Specify the target BAR.

PciAddress: 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 hFPGA, PCIE_BAR PciBar, PCIE_ADDRESS PciAddress, uint8_t Byte);
Parameters: hFPGA: A PCIe handle return by PCIE_Open function. PciBar: Specify the target BAR. PciAddress: 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 hFPGA, PCIE_LOCAL_ADDRESS LocalAddress, void *pBuffer,

uint32_t dwBufSize);
Parameters: hFPGA: 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 hFPGA, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, uint32_t dwDataSize);
Parameters: hFPGA: 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 hFPGA, uint32_t Offset, uint32_t *pData32);
Parameters: hFPGA: A PCIe handle return by PCIE_Open function. Offset: Specify the target byte of offset in PCIe configuration table. pData32: A 4-bytes buffer to retrieve the 32-bit data.
Return Value: Return **true** if read data is successful; otherwise **false** is returned.

3.5 PCIe Reference Design - DDR4

The application reference design shows how to integrated DDR4 Memory Controllers for DDR4-A and DDR4-B DDR4 Memory with the PCIe System and perform 4GB data DMA for both DDR4 Memory. 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:

SystemCD\RTL_Package\demonstration\PCIe_DDR4\demo_batch

The folder includes following files:

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

■ Demonstration Setup

1. Connect 12V DC to FLIK to power on FLIK.
2. Connect the FLIK to your Host PC via a Thunderbolt 3 cable.
3. Configure FPGA with Flik_top.sof by executing the test.bat.
4. Remove and plug the Thunderbolt 3 Cable.
5. Install PCIe driver if necessary.
6. Make sure the Windows has detected the FPGA Board by checking the Windows Control panel.
7. Goto windows_app folder, execute PCIE_DDR4.exe. A menu will appear as shown in **Figure 3-10**.

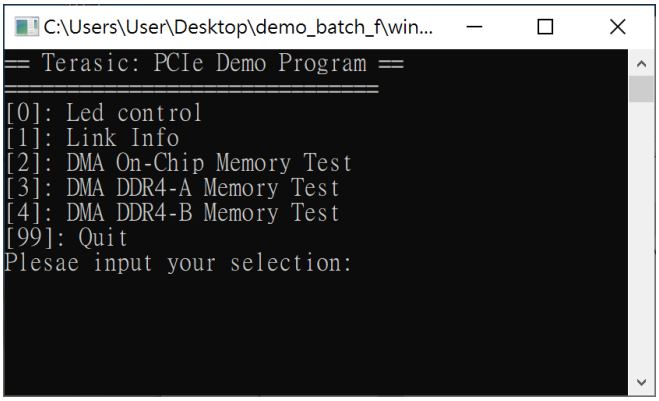


Figure 3-10 Screenshot of Program Menu

8. Type 1 followed by an ENTER key to select Link Info item. The PCIe link information will be shown as in **Figure 3-11**. Gen3 link speed and x4 link width are expected.

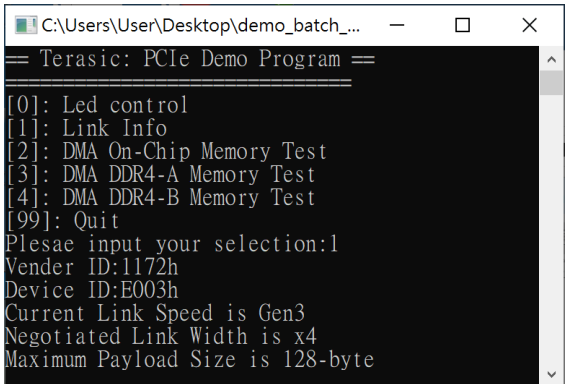
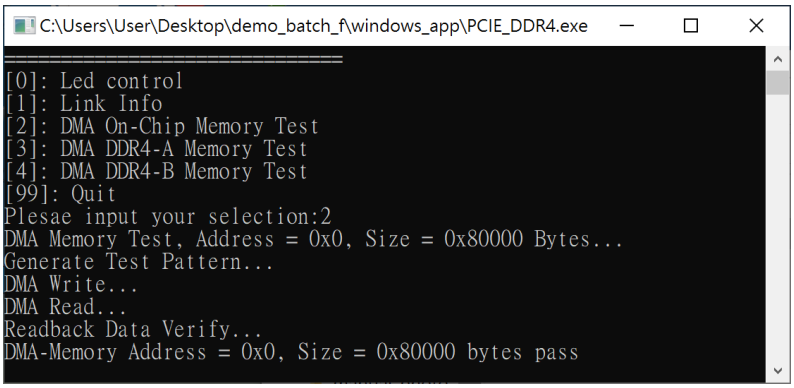


Figure 3-11 Screenshot of Link Info

9. Type 2 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



10. **Figure 3-12**.

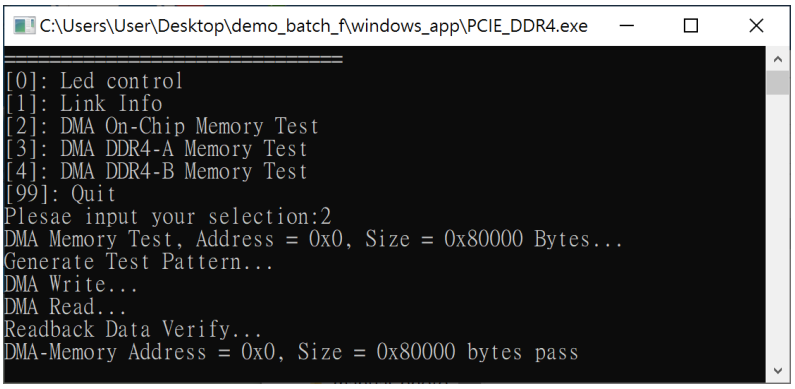
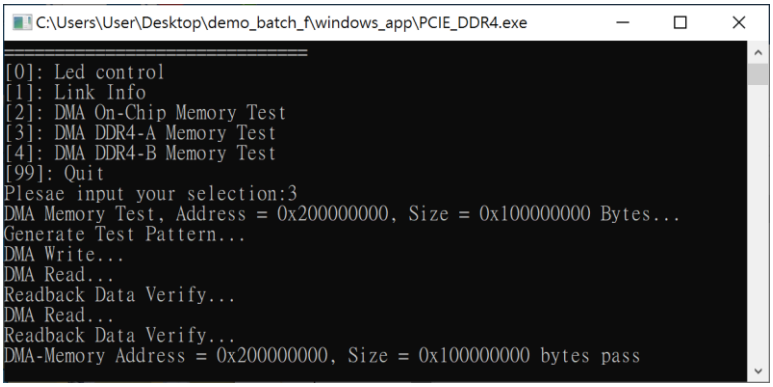


Figure 3-12 Screenshot of On-Chip Memory DMA Test Result

11. Type 3 followed by an ENTER key to select DMA DDR4-A Memory Test item. The DMA write and read test result will be report as shown



12. **Figure 3-13**.

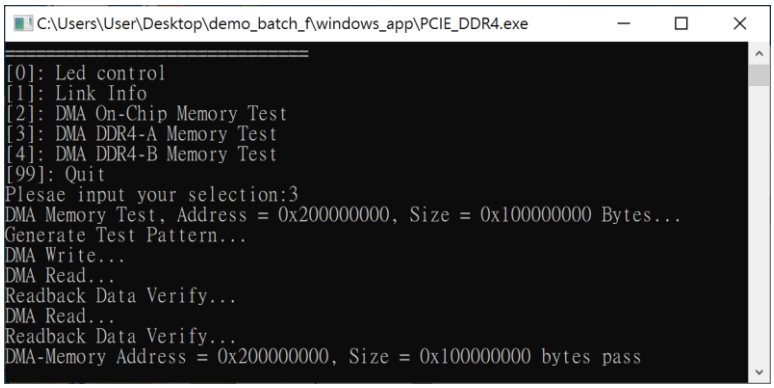


Figure 3-13 Screenshot of DDR4-A Memory DMA Test Result

13. Type 4 followed by an ENTER key to select DMA DDR4-B Memory Test item. The DMA write and read test result will be report as shown in **Figure 3-15.**

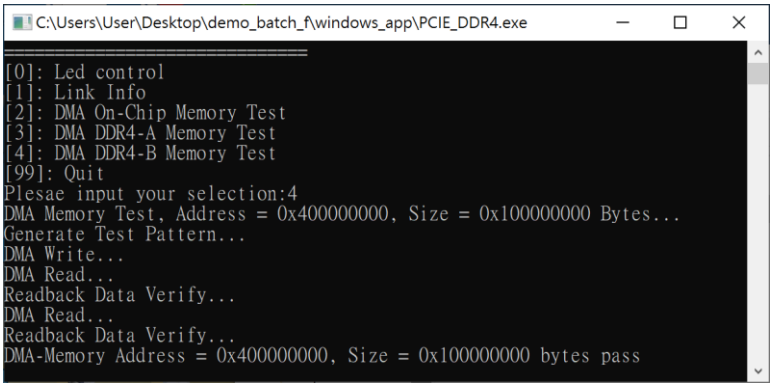


Figure 3-14 Screenshot of DDR4-B Memory DMA Test Result

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

■ Development Tools

- Quartus Prime 19.2 or later Pro Edition
- Visual C++ 2012

■ Demonstration Source Code Location

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

■ FPGA Application Design

Figure 3-15 shows the system block diagram in the FPGA system. In the Qsys, Altera PIO controller is used to control the LED, and the On-Chip memory and DDR4 Memory are used for performing DMA testing. The PIO controllers and the memory are connected to the PCI Express Hard IP controller through the Memory-Mapped

Interface.

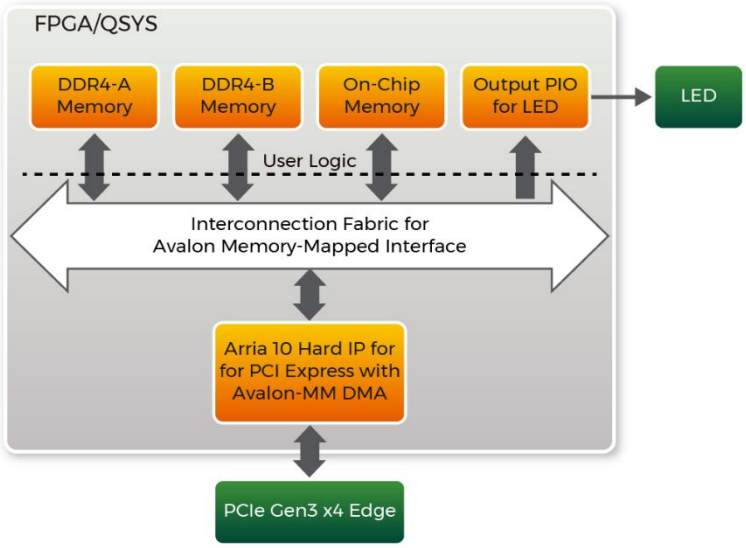


Figure 3-15 Hardware block diagram of the PCIe_DDR4 reference design

■ 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_DDR4.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_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   0x4000010
#define DEMO_PCIE_ONCHIP_MEM_ADDR 0x00000000
#define DEMO_PCIE_DDR4A_MEM_ADDR  0x200000000
#define DEMO_PCIE_DDR4B_MEM_ADDR  0x400000000

#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
#define DMA_MAX_SIZE        (4ull*1024*1024*1024 - 4) //4GB - 4B
#define DMA_CHUNK_SIZE        (2ull*1024*1024*1024) //2GB
```

The base address of LED controllers are 0x4000010 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 DDR4 demo.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the TERASIC_PCIE_AVMM.DLL. Then, it calls 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 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 4

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. Arria 10 Hard IP for PCI Express with Avalon-MM DMA IP is used in this demonstration. For detail about this IP, please refer to Altera document ug_a10_pcie_avmm_dma.pdf.

4.1 PCI Express System Infrastructure

Figure 4-1 shows the infrastructure of the PCI Express System in this demonstration. It consists of two primary components: FPGA System and PC System. The FPGA System is developed based on Arria 10 Hard IP for PCI Express with Avalon-MM DMA. The application software on the PC side is developed by Terasic based on Altera's PCIe kernel mode driver.

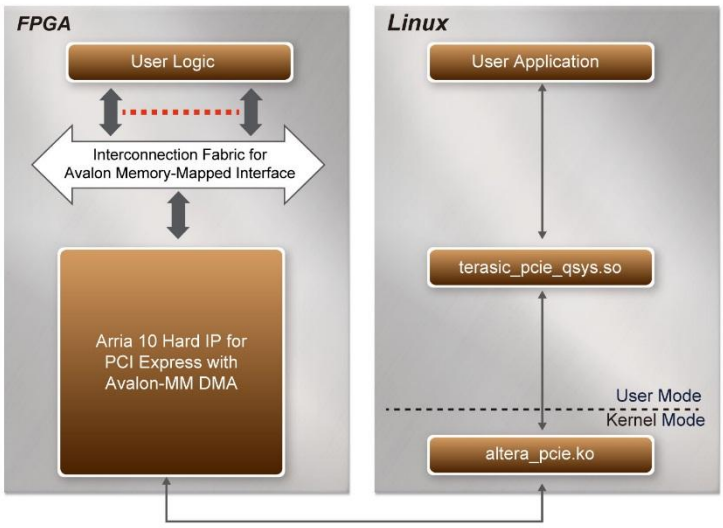


Figure 4-1 PCI Express System Infrastructure

4.2 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 "SystemCD/demonstration/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.

4.3 PCI Express Software Stack

Figure 4-2 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 Intel.

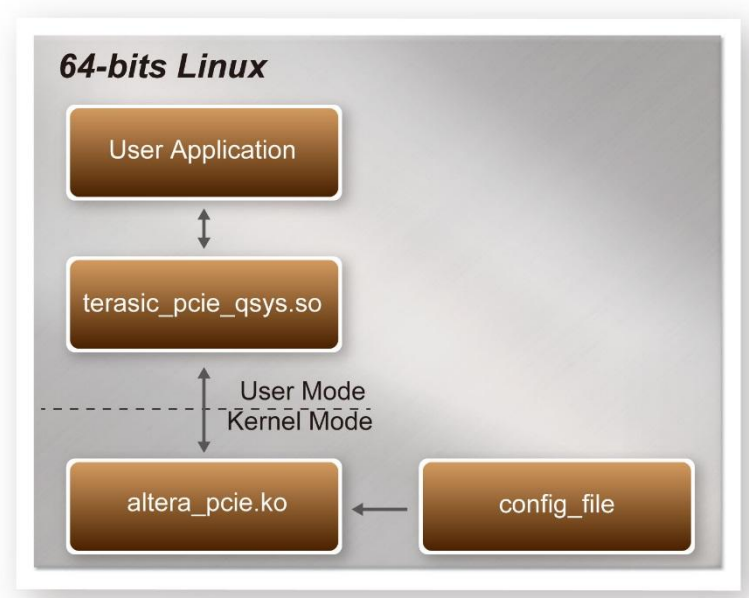


Figure 4-2 PCI Express Software Stack

■ 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:

"SystemCD/demonstration/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. Plug 12V DC to FLIK to power on FLIK.
2. Connect FLIK to Host PC Thunderbolt 3 port by a Thunderbolt 3 cable.
3. Make sure Quartus Programmer and USB-Blaster II driver are installed
4. Open a terminal and use "cd" command to goto the folder "SystemCD/demonstration/PCIe_DDR4/demo_batch".
5. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in

terminal. Replace “/home/user/intelFPGA_pro/19.2/quartus” to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/19.2/quartus
```

6. Execute "sudo -E sh test.sh" command to configure the FPGA.
7. Restart Linux operation system. In Linux, open a terminal and use “cd” command to goto the PCIe_Driver folder.
8. 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 4-3.**
   - make
   - sudo sh load_driver
   - dmesg | tail -n 15

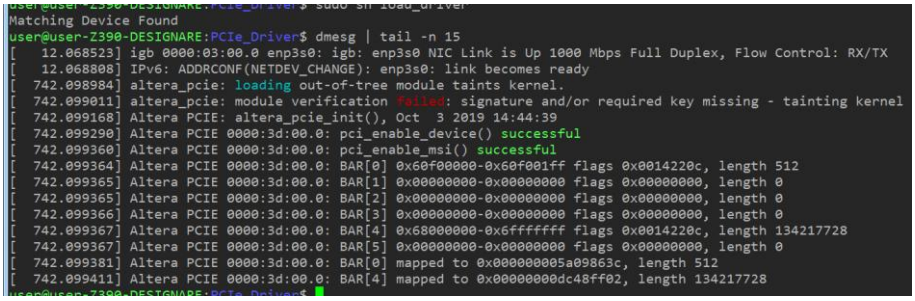


Figure 4-3 Screenshot of install PCIe driver

■ 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 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_qsys.so API. The details of API are described below:

4.4 PCI Express Library API

The API is the same as Windows Library. Please refer to the **Section 3.4** PCI Express Library API in this document.

4.5 PCIe Reference Design - DDR4

The application reference design shows how to add DDR4 Memory Controllers for DDR4-A and DDR4-B into the PCIe Quartus project and perform 4GB data DMA for both DDR4. 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:

SystemCD/RTL_Package/demonstration/PCIe_DDR4/demo_batch

The folder includes following files:

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

■ Demonstration Setup

1. Plug 12V DC to FLIK to power on FLIK.
2. Connect FLIK to Host PC Thunderbolt 3 port by a Thunderbolt 3 cable.
3. Open a terminal and use "cd" command to goto "CDROM/Demonstrations/PCIe_DDR4/demo_batch".
4. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by tying the following commands in terminal. Replace /home/user/intelFPGA_pro/19.2/quartus to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/user/intelFPGA_pro/19.2/quartus
```

5. Execute "sudo -E sh test.sh" command to configure the FPGA
6. Restart Linux
7. Install PCIe driver.

8. Make sure the Linux has detected the FPGA Board.
9. Goto linux_app folder, execute PCIE_DDR4. A menu will appear as shown in **Figure 4-4**.

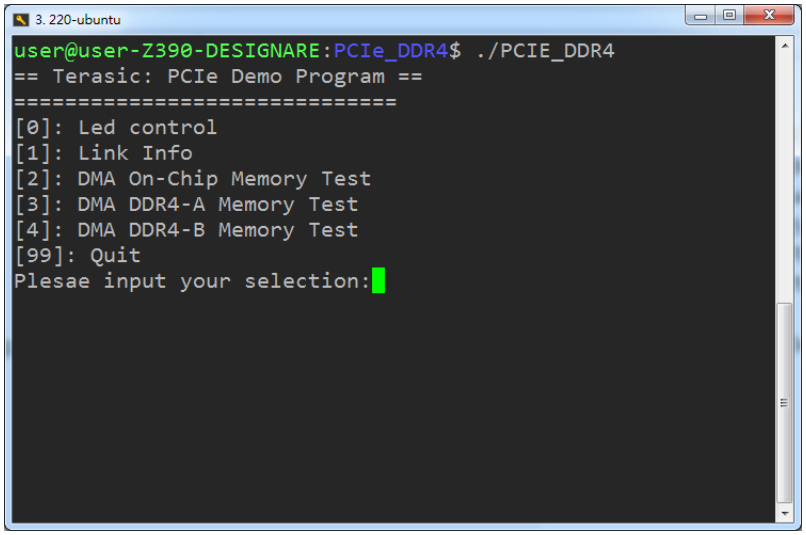
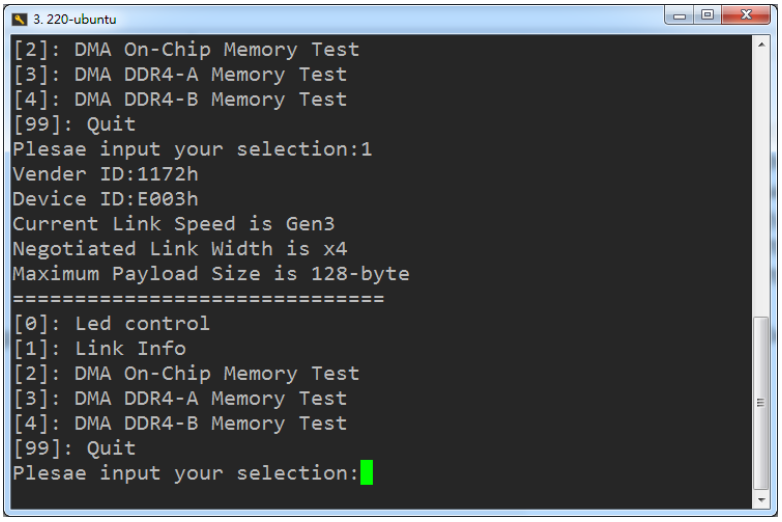


Figure 4-4 Screenshot of Program Menu

10. Type 1 followed by an ENTER key to select Link Info item. The PCIe link information will be shown as in



11. **Figure 4-5**. Gen3 link speed and x8 link width are expected.

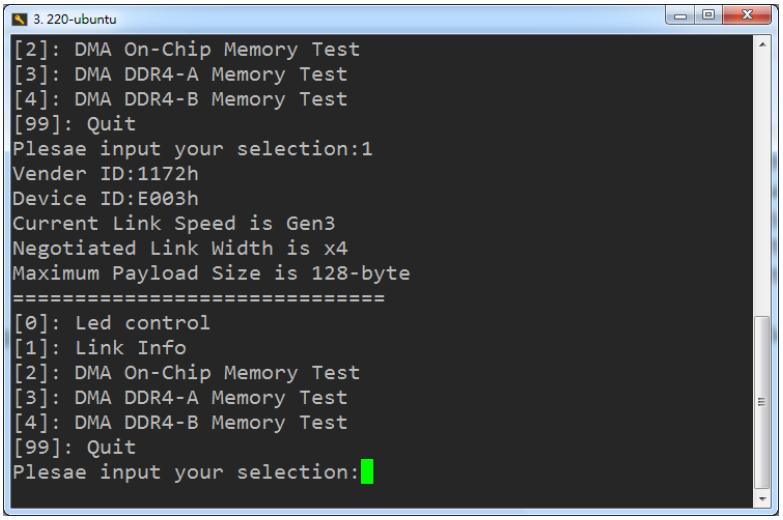


Figure 4-5 Screenshot of Link Info

12. Type 2 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 4-6**.

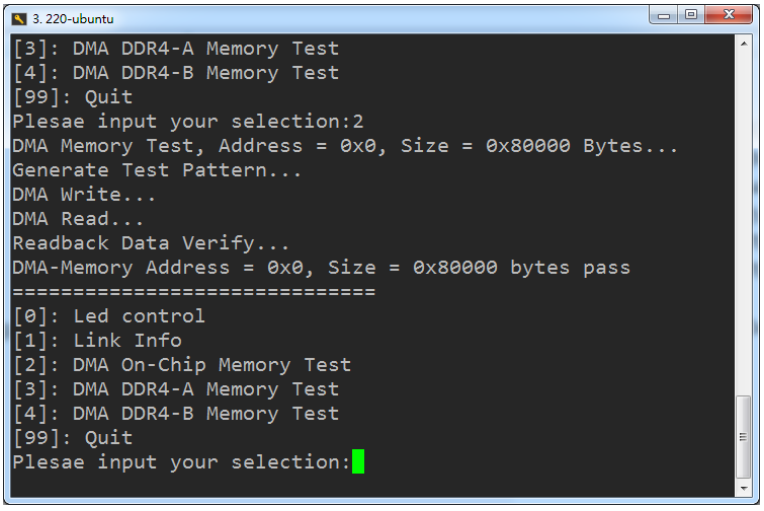


Figure 4-6 Screenshot of On-Chip Memory DMA Test Result

13. Type 3 followed by an ENTER key to select DMA DDR4-A Memory Test item. The DMA write and read test result will be report as shown in **Figure 4-7.**

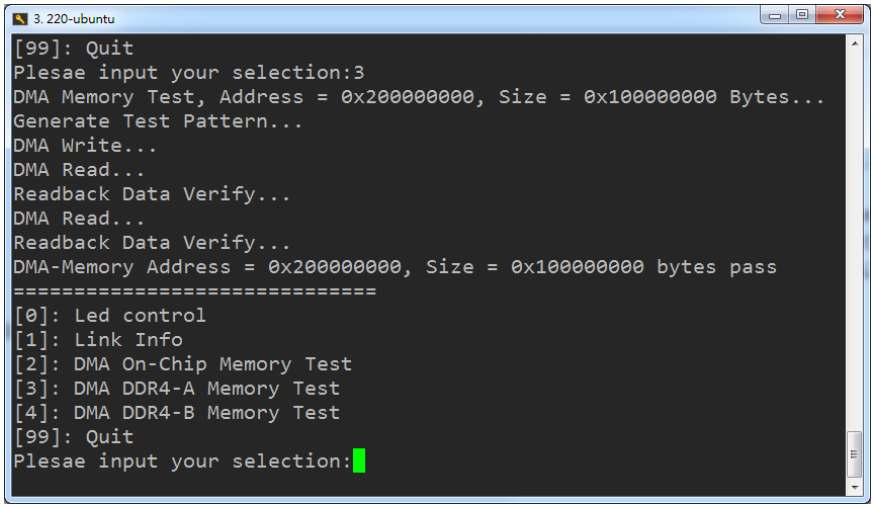


Figure 4-7 Screenshot of DDR4-A Memory DMA Test Result

14. Type 4 followed by an ENTER key to select DMA DDR4-B Memory Test item. The DMA write and read test result will be report as shown in **Figure 4-8**.

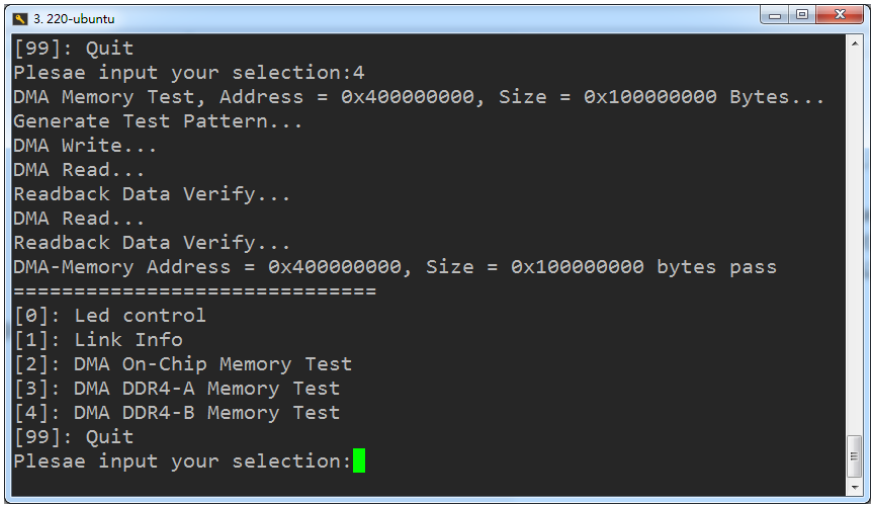


Figure 4-8 Screenshot of DDR4-B Memory DMA Test Result

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

■ Development Tools

- Quartus Prime 19.2 Pro Edition
- GNU Compiler Collection, Version 4.8 is recommended

■ Demonstration Source Code Location

- Quartus Project: RTL_Package/demonstration/PCIE_DDR4
- C++ Project: RTL_Package/demonstration/PCIe_SW_KIT/Linux/PCIe_DDR4

■ FPGA Application Design

Figure 4-9 shows the system block diagram in the FPGA system. In the Qsys, Altera PIO controller is used to control the LED, and the On-Chip memory and DDR4 are used for performing DMA testing. The PIO controllers and the memory are connected to the PCI Express Hard IP controller through the Memory-Mapped Interface.

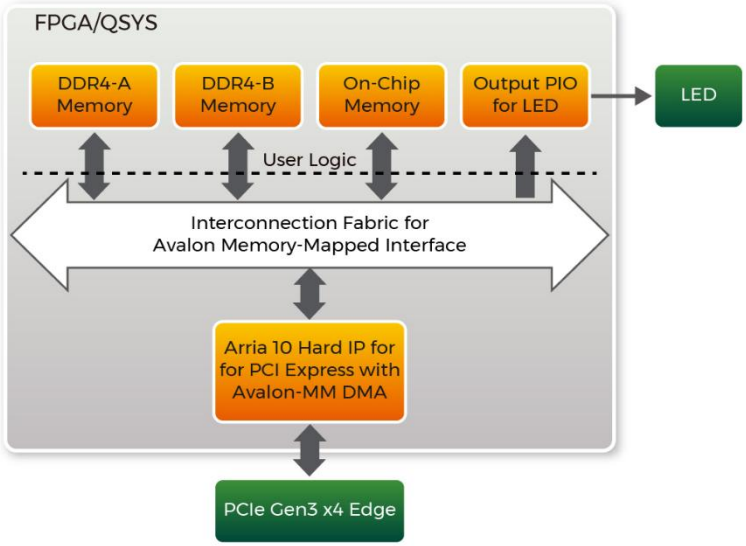


Figure 4-9 Hardware block diagram of the PCIe_DDR4 reference design

■ 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
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_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    0x4000010
#define DEMO_PCIE_ONCHIP_MEM_ADDR 0x00000000
#define DEMO_PCIE_DDR4A_MEM_ADDR  0x200000000
#define DEMO_PCIE_DDR4B_MEM_ADDR  0x400000000

#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
#define DMA_MAX_SIZE         (4ull*1024*1024*1024 - 4) //4GB - 4B
#define DMA_CHUNK_SIZE         (2ull*1024*1024*1024) //2GB
```

The base address of LED controllers is 0x4000010 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 DDR4 demo.

Before accessing the FPGA through PCI Express, the application first calls PCIE_Load to dynamically load the terasic_pcie_qsys.so. Then, it calls 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 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 5

Additional Information

5.1 Getting Help

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

■ Terasic Technologies

9F., No.176, Sec.2, Gongdao 5th Rd,
East Dist, HsinChu City, Taiwan, 30070
Email: support@terasic.com
Web: www.terasic.com
FLIK Web: flik.terasic.com

■ Revision History

Date	Version	Changes
2019.10	First publication	
2020.03	V1.1	Change cover picture