

HAN Pilot Platform

DEMONSTRATION MANUAL

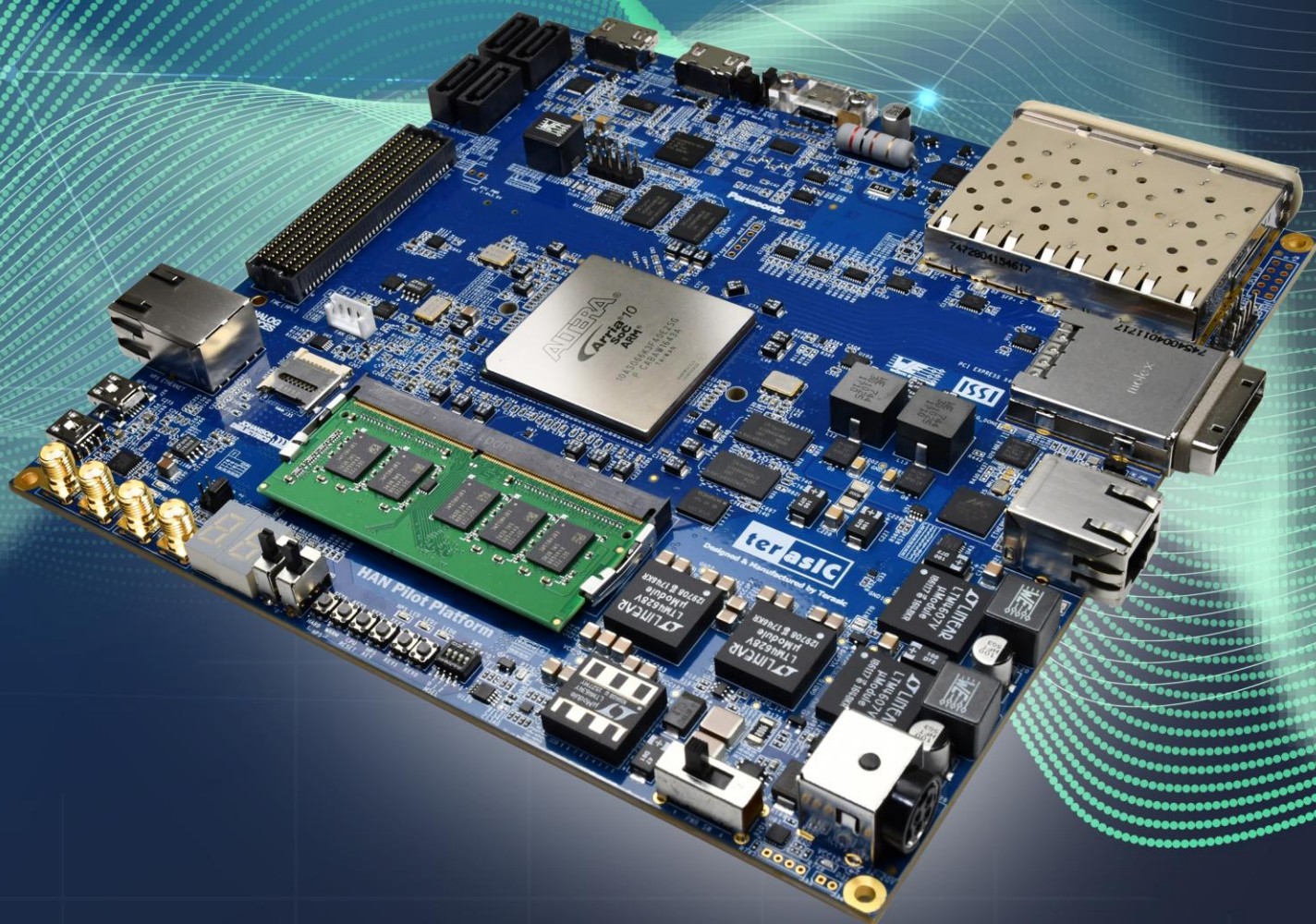


TABLE OF CONTENTS

Chapter 1	Introduction	3
Chapter 2	Examples for FPGA.....	4
2.1	Factory Default Code	4
2.2	Nios II Control for Programmable PLL/ Temperature/ Power/ 9-axis	6
2.3	Nios DDR4 SDRAM Test	12
2.4	RTL DDR4 SDRAM Test.....	14
2.5	USB Type-C DisplayPort Alternate Mode	16
2.6	USB Type-C FX3 Loopback	18
2.7	HDMI TX and RX in 4K Resolution.....	22
2.8	HDMI TX in 4K Resolution	26
2.9	Low Latency Ethernet 10G MAC Demo.....	28
2.10	Socket Server.....	33
2.11	Auto Fan Speed Control.....	39
Chapter 3	Examples for HPS SoC.....	44
3.1	User LED and KEY	44
3.2	Setup USB Wi-Fi Dongle	48
3.3	HPS GPIO Header	51
3.4	Network Socket	55
Chapter 4	Examples for Using both HPS SoC and FPGA	61
4.1	Required Background.....	61
4.2	System Requirements	61
4.3	AXI bridges in Intel SoC FPGA	62
4.4	GHRD Project	63
4.5	Compile and Programming.....	64
4.6	Develop the C Code.....	64
Chapter 5	PCI Express Design for Windows.....	69

5.1 PCI Express System Infrastructure.....	69
5.2 PC PCI Express Software SDK.....	70
5.3 PCI Express Software Stack.....	70
5.4 PCI Express Library API.....	75
5.5 PCIe Reference Design – Fundamental.....	79
5.6 PCIe Reference Design - DDR4.....	85
Chapter 6 PCI Express Design for Linux	92
6.1 PCI Express System Infrastructure.....	92
6.2 PC PCI Express Software SDK.....	93
6.3 PCI Express Software Stack.....	93
6.4 PCI Express Library API.....	96
6.5 PCIe Reference Design – Fundamental.....	100
6.6 PCIe Reference Design - DDR4.....	105
Chapter 7 Linux BSP	112
7.1 Introduction	112
7.2 Use Linux BSP	112
7.3 Linux LXDE VNC Desktop BSP	113
7.4 Linux LXDE HDMI Desktop BSP.....	113
7.5 VNC Desktop OpenCL BSP.....	114

Chapter 1

Introduction

This manual will introduce the various application demonstrations on HAN Pilot Platform. These demonstrations cover most of the interfaces on HAN Pilot Platform. Let users familiarize using these interfaces of the HAN Pilot Platform. Demonstration according to FPGA and HPS Fabrics are divided into three categories:

- Pure use of FPGA fabric resources (Chapter 2)
- Pure use of HPS fabric resources (Chapter 3)
- Use both FPGA and HPS fabric resources (Chapter 4)

In addition, the PCIe example of HAN Pilot Platform will be described separately in one chapter (Chapter 5) because of its high content.

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

- Intel Quartus® Prime Design Software Version 18.0 or later.
- Intel SoC Embedded Design Suite (EDS)

Note: Due to the limitations of the IP version, the DisplayPort and web socket demo needs to be used with Quartus 17.0.

Chapter 2

Examples for FPGA

This chapter provides examples of advance designs implemented by RTL or Qsys on the HAN Pilot Platform. These reference designs cover the features of peripherals connected to the FPGA, such as DDR4, PCIe, HDMI and USB Controller. All the associated files can be found in the directory \Demonstrations\FPGA of HAN Pilot Platform system CD.

2.1 Factory Default Code

The HAN Pilot Platform has a default configuration bit-stream pre-programmed, which demonstrates some of the basic features on board. This demo used LED, 7-Segments, Switch, HDMI transmitter display and fan control.

■ Function Block Diagram

Figure 2-1 shows the function block diagram of this demonstration. This demo used fan controller to control fan and used frame buffer read board picture from ON-CHIP-MEMORY and used scaler scale the picture to 1920x1080 size, display the picture by HDMI TX.

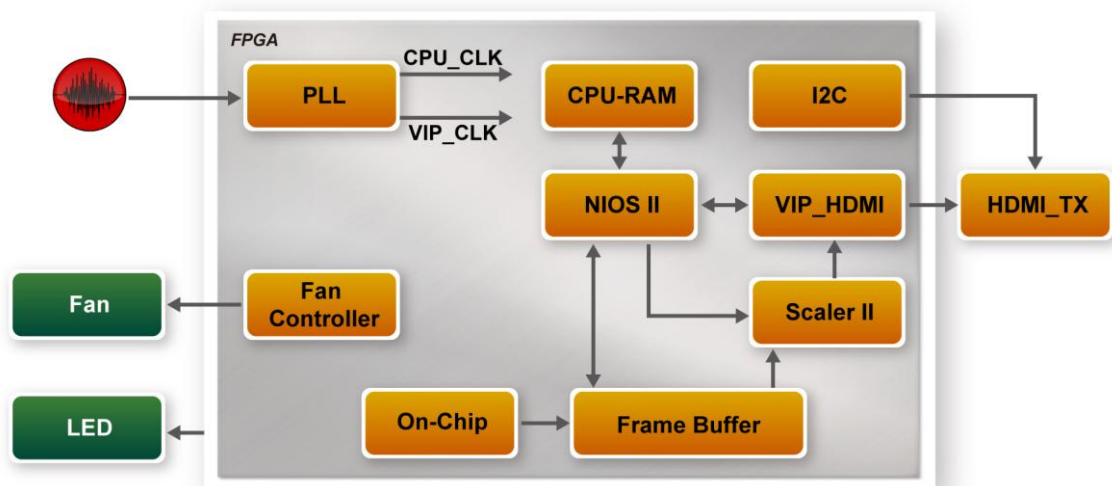


Figure 2-1 Block Diagram of Default Demonstration

■ Design Tools

- Quartus Prime 18.0.0 Standard Edition
- Demonstration Source Code:
 - Project Directory: Demonstration\default_code
 - Bit Stream: default_code.sof or default_code.jic
 - Demonstration Batch File: default_code\demo_batch or default_code\demo_run_batch

NOTE: because the demo included software, and run on on-chip-memory, so use mem_init can package elf to sof, so user can only program .sof or .sof and .elf to run demo.

The demo batch file includes following files:

- Batch File: test.bat
- FPGA Configuration File: default_code.sof or default_code.jic

■ Demonstration Setup

1. Make sure Quartus Prime is installed on the host PC.
2. Connect HAN Pilot Platform to the host PC via USB cable. Install the USB-Blaster II driver if necessary.
3. Connect the HDMI TX to displayer via HDMI cable.
4. Set MSEL[2:0] to 010, set SW0 to 0, SW1 to 0.
5. Power on the HAN Pilot Platform.
6. Execute the demo batch file “test.bat” under the batch file folder \ default_code\demo_batch. You will see the menu as shown in **Figure 2-2**.
7. Select your choice, and program FPGA or program flash.
8. When the demo is running, you can see the LED is blinking and flowing, and the displayer will display the board picture as shown in **Figure 2-3**.

```
C:\WINDOWS\system32\cmd.exe
*****
Makesure MSEL[2:0] is set to "010"
Plesase choose your operation
"1" for programming .sof to FPGA.
"2" for converting .sof to .jic
"3" for programming .jic to EPCQL1024.
"4" for erasing .jic from EPCQL1024.
*****
Please enter your choise: [1,2,3,4]?
```

Figure 2-2 Menu of Default Demonstration

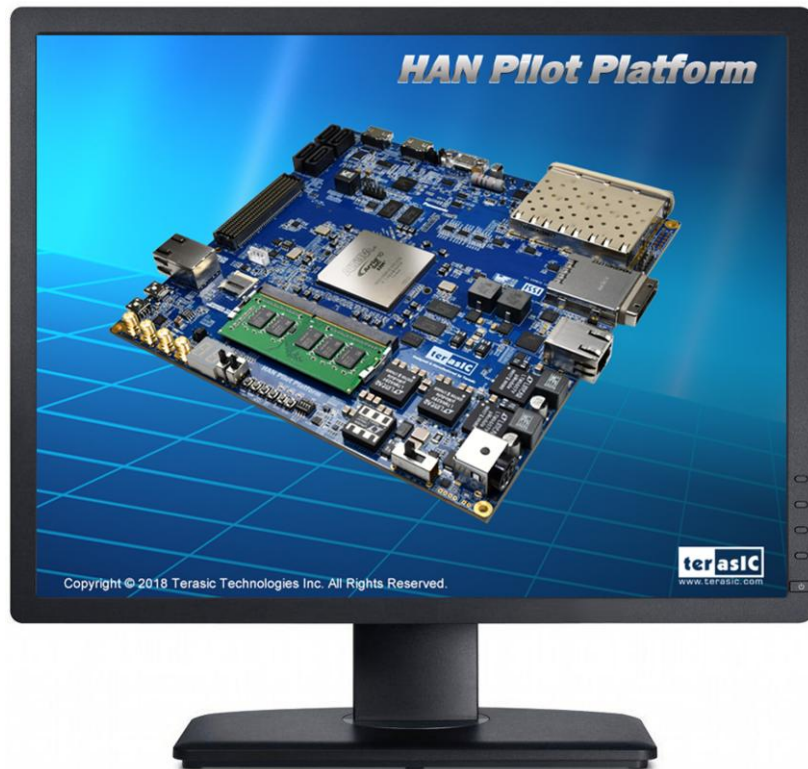


Figure 2-3 Board Picture of Default Demonstration

2.2 Nios II Control for Programmable PLL/ Temperature/ Power/ 9-axis

This demonstration shows how to use the Nios II processor to program two programmable oscillators (CDCM6208 and TXC) on the FPGA board, how to measure the power consumption based on the built-in power measure circuit. The demonstration also includes a function of monitoring system temperature with the on-board temperature sensor, and 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer output with the on-board MPU-9250 Motion Tracking device.

■ System Block Diagram

Figure 2-4 shows the system block diagram of this demonstration. The system requires a 50 MHz clock provided from the board. The five peripherals (including temperature sensor, power monitor, CDCM6208, TXC, and MPU-9250) are all controlled by Nios II through the PIO controller, and all of them are programmed through I2C protocol which is implemented in the C code. The I2C pins from chip are connected to Qsys System Interconnect Fabric through PIO controllers. The Nios II program toggles the PIO controller to implement the I2C protocol. The Nios II program is running in the on-chip memory.

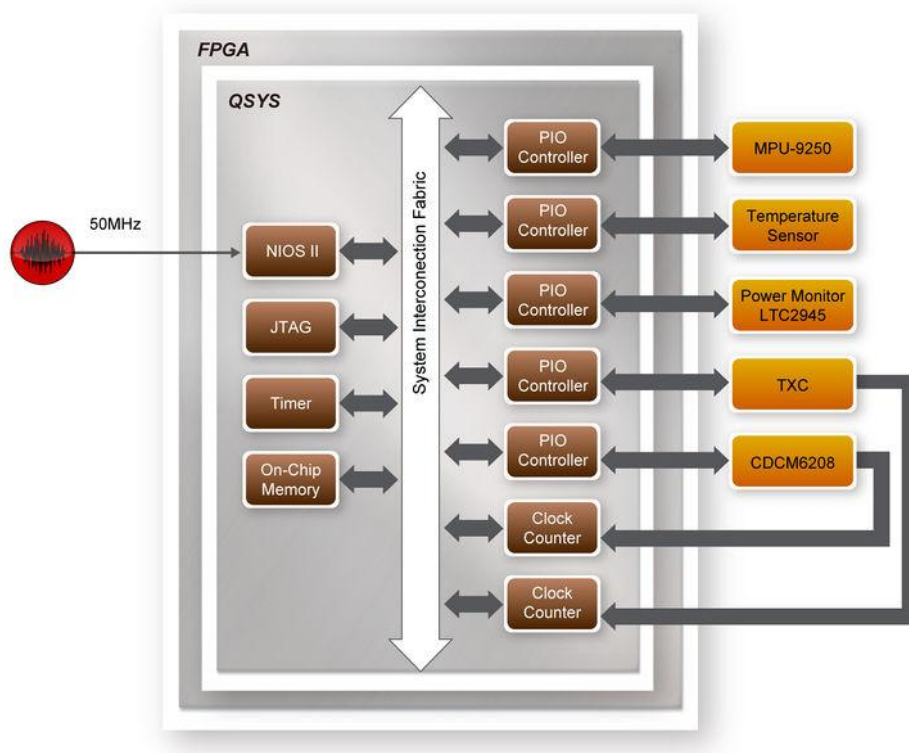


Figure 2-4 Block Diagram of the Nios II Basic Demonstration

The program provides a menu in nios-terminal, as shown in **Figure 2-5** to provide an interactive interface. With the menu, users can perform the test for the temperature sensor, power monitor, external programmable PLL and 9-axis outputs. Note, pressing 'ENTER' should be followed with the choice number.

```

Altera Nios II EDS 18.0 [gcc4]
Using cable "DE10-Advanced [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache <if present>
OK
Downloaded 146KB in 0.2s <730.0KB/s>
Verified OK
Starting processor at address 0x00040244
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE10-Advanced [USB-1]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:

```

Figure 2-5 Menu of Demo Program

In temperature test, the program will display local temperature and remote temperature. The remote temperature is the FPGA temperature, and the local temperature is the board temperature where the temperature sensor located.

A power monitor IC (LTC2945) embedded on the board can monitor Arria10 real-time current and power. This IC can work out current/power value as multiplier and divider are embedded in it. There is a sense resistor R176 (0.006 Ω) for LTC2945 in the circuit, when power on the HAN Pilot Platform, there will be a voltage drop (named Δ SENSE Voltage) on R176. Based on sense resistors, the program of power monitor can calculate the associated voltage, current and power consumption from the LTC2945 through the I2C interface. Please note the device I2C address is 0xD4.

The MPU-9250 consists of two dies, one die houses the 3-axis gyroscope and 3-axis accelerometer, and the other die houses the a-axis magnetometer. Similarly, the MPU-9250 provides complete 9-axis output through the I2C interface.

In the external PLL programming test, the program will program the PLL first, and subsequently will use Terasic QSYS custom CLOCK_COUNTER IP to count the clock count in a specified period to check whether the output frequency is changed as configured. For CDCM6208 programming, the program can control the CDCM6208 to configure the output frequency of SATA/PCIE/DDR4A/DDR4B/DDR4H REFCLK according to your choice. Please note the device I2C address is 0xA8. For TXC programming, the program can control the TXC to configure the output frequency of HDMI/SFP+/FMC/DP/USB REFCLK according to your choice. There are five TXC's ICs for clock generator, divided into two group, TXCA and TXCB. The HDMI and SFP+ reference clock generators share the same I2C bus, REFCLK0_SCL/REFCLK0_SDA, and are grouped into TXCA. The FMC, DP, and USB reference clock generators also share the same I2C bus, REFCLK1_SCL/REFCLK1_SDA, and are grouped into TXCB.

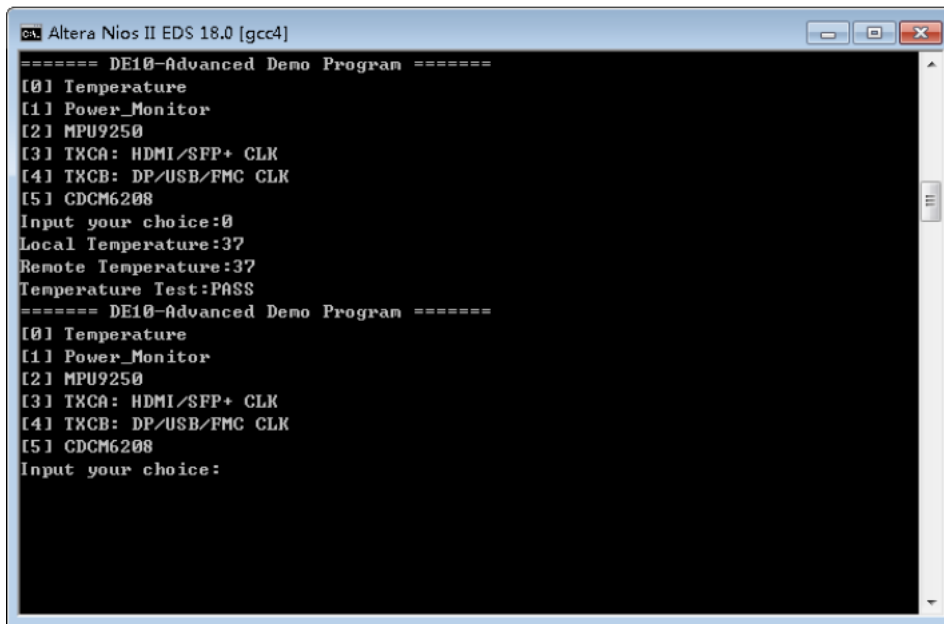
■ Demonstration File Location

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

■ Demonstration Setup and Instructions

1. Make sure Quartus Prime is installed on the host PC.
2. Set MSEL[2:0] to 010
3. Power on the FPGA board.
4. Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
5. Execute the demo batch file “test.bat” under the batch file folder: Basic_Demo\demo_batch.
6. After the Nios II program is downloaded and executed successfully, a prompt message will be displayed in nios2-terminal.

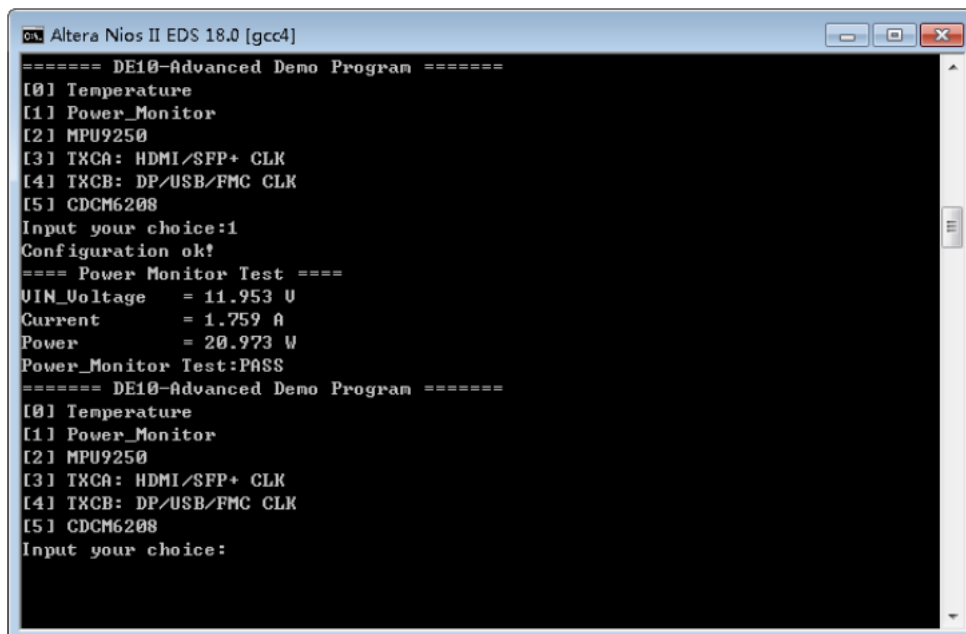
7. For temperature test, please input key '0' and press 'Enter' in the nios-terminal, as shown in [Figure 2-6](#).



```
Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:0
Local Temperature:37
Remote Temperature:37
Temperature Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:
```

Figure 2-6 Temperature Demo

8. For power monitor test, please input key '1' and press 'Enter' in the nios-terminal, the Nios II console will display the values of voltage, current and power as shown in [Figure 2-7](#).



```
Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:1
Configuration ok!
==== Power Monitor Test ====
VIN_Voltage   = 11.953 V
Current       = 1.759 A
Power         = 20.973 W
Power_Monitor Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:
```

Figure 2-7 Power Monitor Demo

9. For 9-axis test, please input key '2' and press 'Enter' in the nios-terminal, the Nios II console will display the values of 9-axis as shown in [Figure 2-8](#).

```
Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:2
9-axis info
ax = 0.215, ay = 0.211, az = 10.415
gx = 0.015, gy = 0.014, gz = 0.028
mx = 15.290, my = 44.645, mz = 43.186
MPU9250 Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:
```

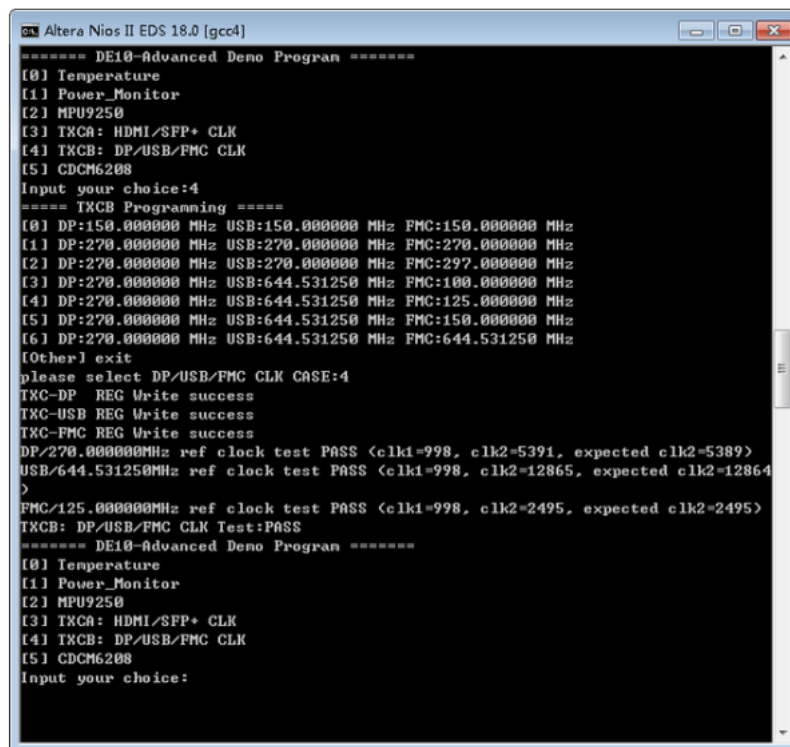
Figure 2-8 MPU-9250 Demo

10. For programmable PLL TXCA test, please input key '3' and press 'Enter' in the nios-terminal first, then select the desired output frequency of HDMI/SFP+ REFCLK, as shown in **Figure 2-9**.

```
Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:3
===== TXCA Programming =====
[0] HDMI:100.000000 MHz SFP:644.531250 MHz
[1] HDMI:135.000000 MHz SFP:644.531250 MHz
[2] HDMI:148.500000 MHz SFP:644.531250 MHz
[3] HDMI:300.000000 MHz SFP:644.531250 MHz
[4] HDMI:600.000000 MHz SFP:644.531250 MHz
[Other] exit
please select HDMI/SFP CLK CASE:2
TXC-HDMI REG Write success
TXC-SFP REG Write success
HDMI/148.500MHz ref clock test PASS <clk1=998, clk2=2965, expected clk2=2964>
SFP/644.531MHz ref clock test PASS <clk1=998, clk2=12865, expected clk2=12864>
TXCA: HDMI/SFP+ CLK Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:
```

Figure 2-9 TXCA Demo

11. For programmable PLL TXCB test, please input key '4' and press 'Enter' in the nios-terminal first, then select the desired output frequency of DP/USB/FMC REFCLK, as shown in **Figure 2-10**.



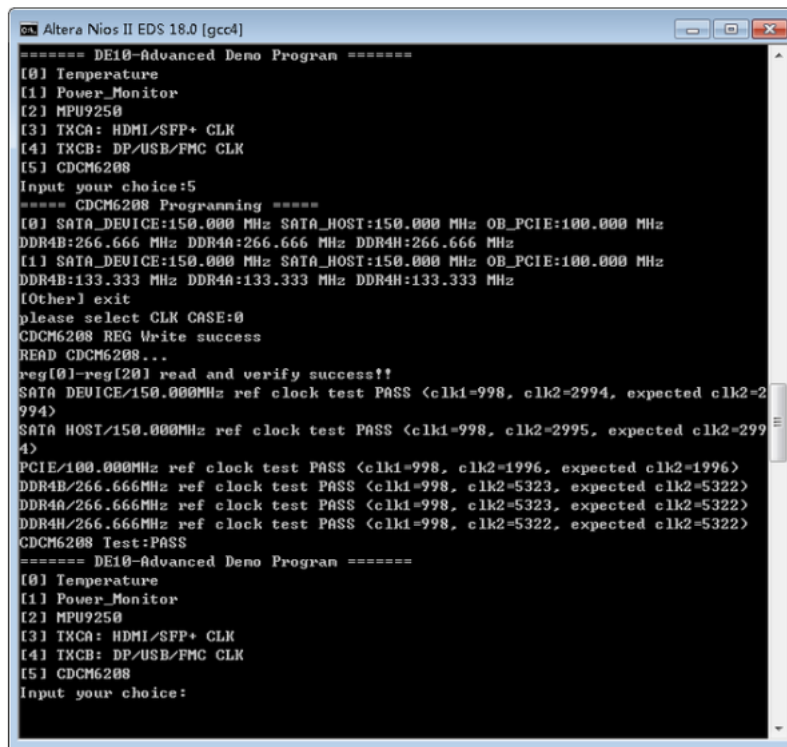
```

Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:4
===== TXCB Programming =====
[0] DP:150.000000 MHz USB:150.000000 MHz FMC:150.000000 MHz
[1] DP:270.000000 MHz USB:270.000000 MHz FMC:270.000000 MHz
[2] DP:270.000000 MHz USB:270.000000 MHz FMC:297.000000 MHz
[3] DP:270.000000 MHz USB:644.531250 MHz FMC:100.000000 MHz
[4] DP:270.000000 MHz USB:644.531250 MHz FMC:125.000000 MHz
[5] DP:270.000000 MHz USB:644.531250 MHz FMC:150.000000 MHz
[6] DP:270.000000 MHz USB:644.531250 MHz FMC:644.531250 MHz
[Other] exit
please select DP/USB/FMC CLK CASE:4
TXC-DP REG Write success
TXC-USB REG Write success
TXC-FMC REG Write success
DP/270.000000MHz ref clock test PASS <clk1=998, clk2=5391, expected clk2=5389>
USB/644.531250MHz ref clock test PASS <clk1=998, clk2=12865, expected clk2=12864>
FMC/125.000000MHz ref clock test PASS <clk1=998, clk2=2495, expected clk2=2495>
TXCB: DP/USB/FMC CLK Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:

```

Figure 2-10 TXCB Demo

12. For programmable PLL CDCM6208 test, please input key '5' and press 'Enter' in the nios-terminal first, then select the desired output frequency of SATA/ PCIE/ DDR4A/ DDR4B/ DDR4H REFCLK, as shown in [Figure 2-11](#).



```
Altera Nios II EDS 18.0 [gcc4]
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:5
===== CDCM6208 Programming =====
[0] SATA_DEVICE:150.000 MHz SATA_HOST:150.000 MHz OB_PCIE:100.000 MHz
DDR4B:266.666 MHz DDR4A:266.666 MHz DDR4H:266.666 MHz
[1] SATA_DEVICE:150.000 MHz SATA_HOST:150.000 MHz OB_PCIE:100.000 MHz
DDR4B:133.333 MHz DDR4A:133.333 MHz DDR4H:133.333 MHz
[other] exit
please select CLK CASE:0
CDCM6208 REG Write success
READ CDCM6208...
reg[0]-reg[20] read and verify success!!
SATA_DEVICE/150.000MHz ref clock test PASS <clk1=998, clk2=2994, expected clk2=2994>
SATA_HOST/150.000MHz ref clock test PASS <clk1=998, clk2=2995, expected clk2=2994>
PCIE/100.000MHz ref clock test PASS <clk1=998, clk2=1996, expected clk2=1996>
DDR4B/266.666MHz ref clock test PASS <clk1=998, clk2=5323, expected clk2=5322>
DDR4A/266.666MHz ref clock test PASS <clk1=998, clk2=5323, expected clk2=5322>
DDR4H/266.666MHz ref clock test PASS <clk1=998, clk2=5322, expected clk2=5322>
CDCM6208 Test:PASS
===== DE10-Advanced Demo Program =====
[0] Temperature
[1] Power_Monitor
[2] MPU9250
[3] TXCA: HDMI/SFP+ CLK
[4] TXCB: DP/USB/FMC CLK
[5] CDCM6208
Input your choice:
```

Figure 2-11 CDCM6208 Demo

2.3 Nios DDR4 SDRAM Test

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 QSYS. We describe how the Altera’s “Arria 10 External Memory Interfaces” IP is used to access the two DDR4-Sodimm on the FPGA board, and how the Nios II processor is used to read and write the SDRAM for hardware verification. The DDR4 SDRAM controller handles the complex aspects of using DDR4 SDRAM by initializing the memory devices, managing SDRAM banks, and keeping the devices refreshed at appropriate intervals.

■ System Block Diagram

Figure 2-12 shows the system block diagram of this demonstration. The QSYS system requires one 50 MHz and two 266.667MHz clock source. The two 266.667 MHz clock source is provided by CDCM6208 clock generator on the board. The 50MHz is used by IO PLL to generate 200MHz for Nios Processor and On-chip Memory. The two 266.667MHz clock are used as reference clocks for the DDR4 controllers. There are two DDR4 Controllers are used in the demonstrations. Each controller is responsible for one DDR4 SDRAM. Each DDR4 controller is configured as a 1 GB DDR4-1066MHz controller. The DDR4A controllers are designed as 1GB rather 4GB is due to address space limitation of Nios II processor. Nios II processor is used to perform memory test. The Nios II program is running in the On-Chip Memory. A PIO Controller is used to monitor buttons status which is used to trigger starting memory testing.

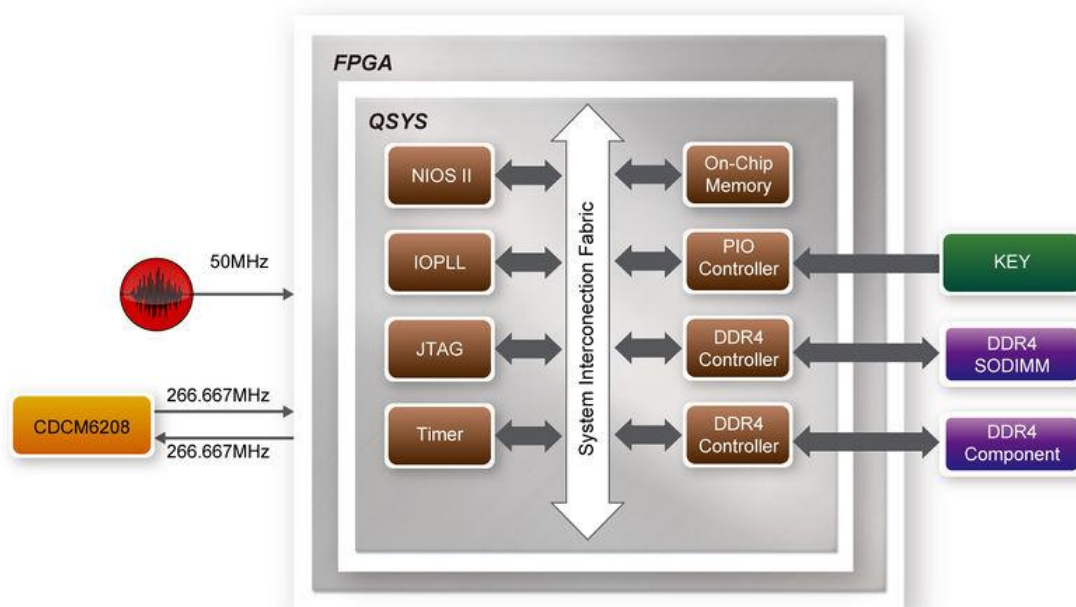


Figure 2-12 Block Diagram of the DDR4 Basic Demonstration

The system flow is controlled by a Nios II program. First, the Nios II program writes test patterns into the whole 1 GB of SDRAM. Then, it calls Nios II system function (alt_dache_flush_all) to make sure all data has been written to SDRAM. Finally, it reads data from SDRAM for data verification. The program will show progress in JTAG-Terminal when writing/reading data to/from the SDRAM. When verification process is completed, the result is displayed in the JTAG-Terminal.

■ Design Tools

- Quartus Prime 18.0.0 Standard Edition
- Nios II Software Build Tools for Eclipse 18.0

■ Demonstration Source Code

- Quartus Project directory: NIOS_DDR4
- Nios II Eclipse: NIOS_DDR4 \software
- Nios II Project Compilation

■ Nios II Project Compilation

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

■ Demonstration Batch File

Demo Batch File Folder: NIOS_DDR4 \demo_batch

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat, test.sh
- FPGA Configure File: NIOS_DDR4.sof

- Nios II Program: DDR4_Test.elf

■ Demonstration Setup

Please follow below procedures to setup the demonstration.

1. Make sure Quartus Prime and Nios II are installed on your PC.
2. Make sure DDR4 SODIMM are installed on the FPGA board.
3. Set MSEL[2:0] to 010.
4. Power on the FPGA board.
5. Use USB Cable to connect PC and the FPGA board and install USB Blaster II driver if necessary.
6. Execute the demo batch file “test.bat” under the folder “NIO5_DDR4\demo_batch”.
7. After Nios II program is downloaded and executed successfully, a prompt message will be displayed in nios2-terminal.
8. Press Key0~Key1 of the FPGA board to start SDRAM verify process. Press Key0 for continued test.
9. The program will display progressing and result information, as shown in **Figure 2-13**.

```

Altera Nios II EDS 18.0 [gcc4]
Using cable "DE10-Advanced [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x81020244
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE10-Advanced [USB-1]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

==== DDR4 Test! Size=A:1024MB/B:1024MB====

Press any BUTTON on the board to start test [BUTTON0 for continued test]
====> DDR4 Testing, Iteration: 1
A/B Calibration Duration:0.721 seconds, status=5h
== DDR4-A Testing...
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
DDR4 test:Pass, 138 seconds
== DDR4-B Testing...
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
DDR4 test:Pass, 95 seconds

Press any BUTTON on the board to start test [BUTTON0 for continued test]

```

Figure 2-13 Progress and Result Information for the DDR4 Demonstration

2.4 RTL DDR4 SDRAM Test

This demonstration performs a memory test function on the one DDR4 SO-DIMM (DDR4A) and one DDR4 Component (DDR4B) on the HAN Pilot Platform. The memory size of DDR4 SO-DIMM is 4GB and DDR4 Component is 1GB.

■ Function Block Diagram

Figure 2-14 shows the function block diagram of this demonstration. There are two DDR4 SDRAM controllers. The controller uses 266.667 MHz as a reference clock. It generates one 1066MHz clock as memory clock from the FPGA to the memory and the controller itself runs at quarter-rate in the FPGA i.e. 266.667 MHz.

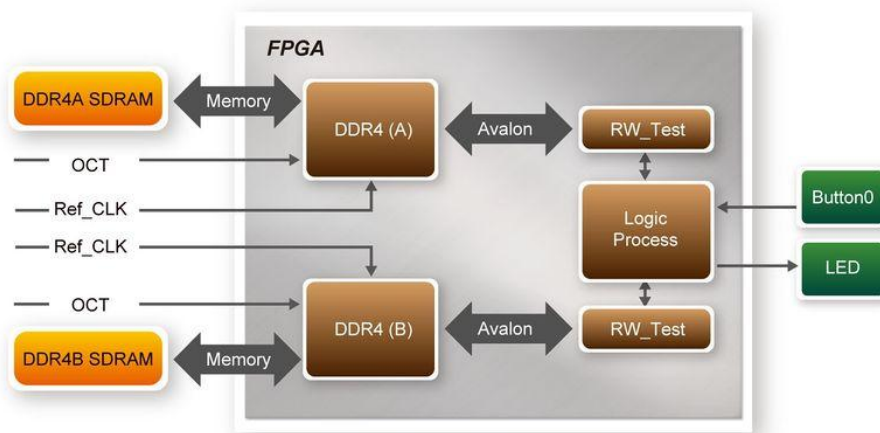


Figure 2-14 Block Diagram of DDR4 x2 Demonstration

■ Design Tools

- Quartus Prime 18.0.0 Standard Edition
- Demonstration Source Code:
 - Project Directory: Demonstration\RTL_DDR4
 - Bit Stream: RTL_DDR4.sof

■ Demonstration Batch File

Demo Batch File Folder: RTL_DDR4 \demo_batch

The demo batch file includes following files:

- Batch File: test.bat
- FPGA Configuration File: RTL_DDR4.sof

■ Demonstration Setup

1. Make sure Quartus Prime is installed on the host PC.
2. Connect HAN Pilot Platform to the host PC via USB cable. Install the USB-Blaster II driver if necessary.
3. Set MSEL[2:0] to 010.
4. Power on the HAN Pilot Platform.
5. Execute the demo batch file “test.bat” under the batch file folder \ RTL_DDR4\demo_batch.
6. Press KEY0 on HAN Pilot Platform to start the verification process. When KEY0 is released, LED0, LED1 should start blinking. After approximately 2 seconds, LED1 and LED2 should stop blinking and stay on to indicate the DDR4 (A) and DDR4 (B) have passed the test, respectively. **Table 2-1** lists the LED indicators.
7. If LED0 or LED1 does not start blinking upon releasing KEY0, it indicates local_cal_success of

the corresponding DDR4 fails.

8. If LED0 or LED1 fail to remain on after 2 seconds, the corresponding DDR4 test has failed.
9. Press KEY0 again to regenerate the test control signals for a repeat test.

Table 2-1 LED Indicators

Name	Description
LED0	DDR4 (A) test result
LED1	DDR4 (B) test result

2.5 USB Type-C DisplayPort Alternate Mode

This section introduces how to implement a DisplayPort Source based on USB Type-C DisplayPort Alternate Mode. The demo includes two major parts: DisplayPort and USB Type-C.

For DisplayPort design, DisplayPort Intel FPGA IP is used to generate DisplayPort TX video. The DisplayPort design is refer to the document :Arria 10 DisplayPort Design Example using on board connector (TX Only).

For USB Type-C, system need to monitor the information sent from the USB Type-C Port Controller CYPD3125 (EZ-PD CCG3). From the sent information, system can know whether the plug-in device is a DisplayPort monitor and the DP lane number is 4 or 2, and system have to configure the DisplayPort crossbar switch so the FPGA transceiver signals can be routed to the type-c port correctly.

The Quartus Project USBC_DP_4K is designed for 4K monitor, and USBC_DP_FullHD is design for Full HD Monitor. If your Type-C monitor only supports Full HD, please use the USB_DP_FullHD for the demo setup.

■ System Block Diagram

Figure 2-15 shows the system block diagram for the DisplayPort Demo. When a Type-C monitor is plugged into the Type-C Connector, the Type-C Port Controller (EZ-PD CCG3) will enable 5V power for the monitor. When a Type-C device is plug-in or removed from the Type-C connector, the CCG3 will notify FPGA through the IC2 interface. CCG3 will sends one byte data to 0 offset address in the I2C Slave Port. The meaning of the data is shown in **Table 2-2**. If attached device is a DisplayPort monitor, then DisplayPort crossbar switch is configured so the transceivers signals are routed to the type-c connector correctly.

For DisplayPort design, the Hot Plug Detect (HPD) causes the DisplayPort source to initialize the link via AUX channel. The DisplayPort IP generates parallel Video data and FPGA transceivers are used to serial the video data. For 4K video data, 4 TX transceivers are used with reference clock 270 MHz. For Full HD video data, 4 or 2 TX transceivers are used. The input video data for DisplayPort IP is generated by VIP Test Pattern Generator II IP and VIP Clocked Video Output II IP.

In the system, a Nios II processor is used to control the DisplayPort IP. The Nios II Processor is running on on-chip memory with 100Mhz.

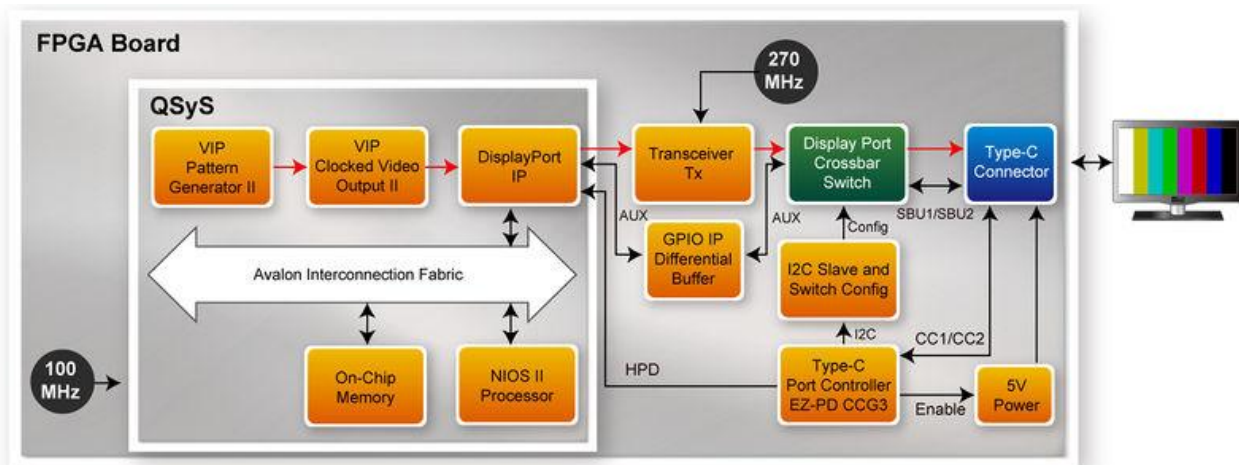


Figure 2-15 Block Diagram of DisplayPort Demo

Table 2-2 Information sent by CCG3

Data[6:5]	Description
0	No Device is attached
1	Only USB 3.1 Device is attached
2	4 Lane DisplayPort monitor is attached
3	USB and 2 Lane DisplayPort monitor is attached

Data[4]	Description
0	Cable is normal
1	Cable is flip

■ Demonstration File Locations

- For 4K Video Output:
 - Hardware project directory: USBC_DP_4K
 - Bitstream used: USBC_DP_golden_top.sof
 - Nios II Program: dp_demo_test.elf
 - Demo batch file: USBC_DP_4K\demo_batch\test.bat
- For Full-HD Video Output:
 - Hardware project directory: USBC_DP_FullHD
 - Bitstream used: USBC_DP.sof.sof
 - Nios II Program: dp_demo_test.elf
 - Demo batch file: USBC_DP_FullHD\demo_batch\test.bat

Note: Quartus 17.0 Standard is used for these demonstration design. Use Quartus 18.0 could make malfunction.

■ Demonstration Setup and Instructions

1. Make sure Quartus Prime is installed on your PC.

2. Set MSEL[2:0] to 010.
3. Power on the FPGA board.
4. Use the Mini USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
5. Execute test.bat under the demo_batch folder.
6. Connect USB Type-C Connector (J2) to a 4K Type-C Monitor (or Full HD Type-C Monitor) via a Type-C Cable.
7. You will see a color bar pattern on the Type-C Monitor.

2.6 USB Type-C 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 type-C connector onto HAN Pilot Platform, a type-C cable is reversible for plugging in the USB type-C connector. This demonstration also implements the auto-switching mechanism for a type-C cable plugging in on either side.

■ Function Block Diagram

Figure 2-16 shows the function block diagram of the USBC_FX3 demonstration. This design comprises two parts, USB3.0/USB2.0 TX&RX crossbar switch controlling and FX3 data transferring. As the Type-C connector is reversible, a Type-C port Controller (CYPD3125) is required for USB3.0/USB2.0 TX&RX crossbar switch controlling. When the Type-C cable is plugged in, the CC (Configuration Channel) signal is communicating with the controller. The controller then transfers the data to the Slave IC (FPGA) by I2C in Master mode, and There will be a slave I2C module in the FPGA to decode the signal and send the control signal to control the RX/TX direction of the PI3USB31532 USB3.0 signal. Here is a LED0 on the board to indicate the cable plugging direction, and LED1 indicates whether there is a USB connection signal or not. 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:

USB_AUTO_DETECT: This module can decode the I2C signal from the Type-C Port Controller (CYPD3125 IC), then timely switch the RX/TX direction of Type-C port (by controlling the PI3USB31532 IC) and control the USB 3.0 Mux/Demux9 (HD3SS3212) switching to transfer the USB 3.0 signal to FX3 module. As shown in Figure 4-1, the LED0 indicates the Type-C connector RX/TX direction, the LED1 indicates the USB 3.0 signal input.

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.

KEY0: It is used to reset FX3 module.

PMODE[2:0]: The HAN Pilot 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.

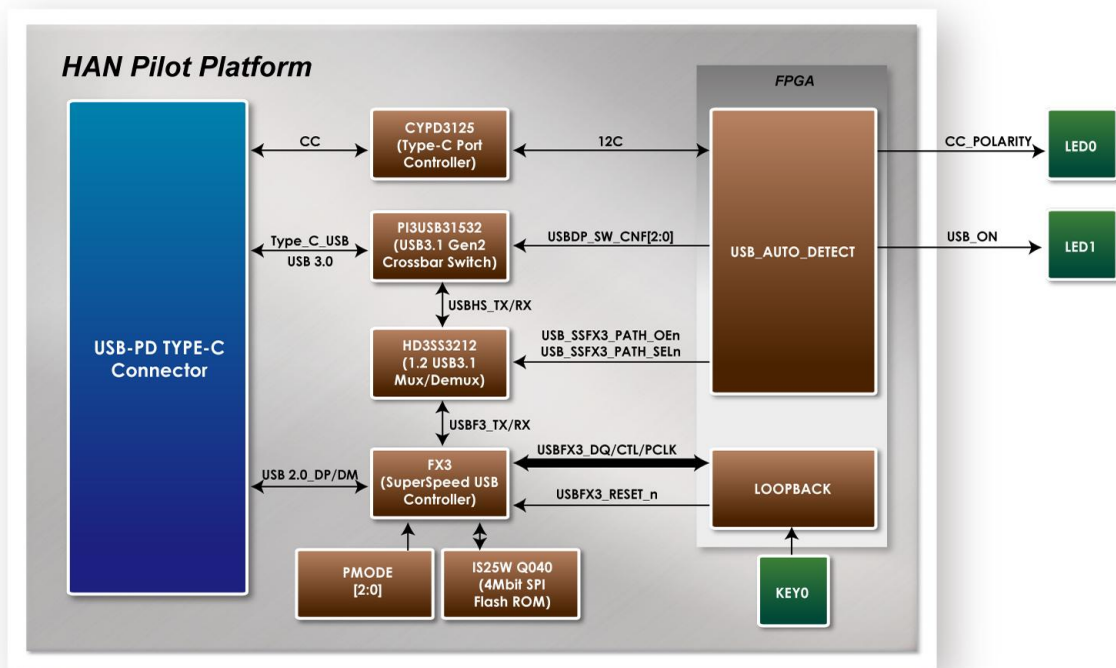


Figure 2-16 Block diagram of the USBC FX3 design

■ Demonstration Setup

- Hardware Setting Up, as shown in Figure 2-17.

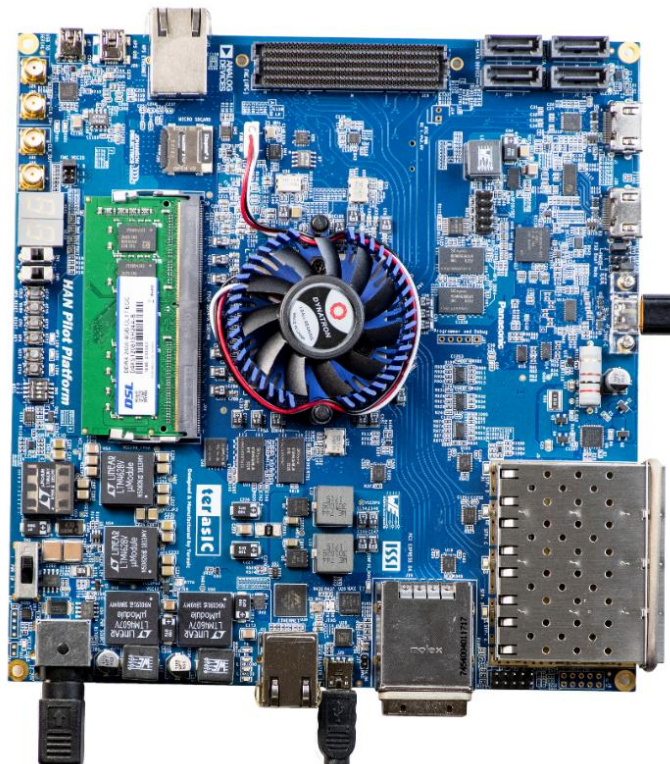


Figure 2-17 USBC FX3 Demo Hardware Setting Up

- Design Tools
 - Quartus Prime 18.0 Standard Edition

- Demonstration Source Code
 - Quartus project directory: USBC_FX3
 - Bitstream used: USBC_FX3.sof
- Demonstration Batch File
 - Demo batch file folder: demonstrations\USBC_FX3\demo_batch
- Demonstration Setup
 - Connect the HAN Pilot Platform USB Blaster II connector (J20) to the host PC with a USB cable and install the USB-Blaster II driver if necessary.
 - Use a Type-C cable to connect the HAN Pilot Platform and a PC (with a Type-C connector). As shown in **Figure 2-17**.
 - Plug the 12V adapter to HAN Pilot Platform DC 12V power connector (J28).
 - Set MSEL[2:0] to 010.
 - Power on the HAN Pilot Platform.
 - Execute the demo batch file “test.bat” from the directory \FPGA\USBC_FX3\demo_batch.
 - Install the FX3 driver: the driver for Windows 7 is in the \FPGA\USBC_FX3\demo_batch\Driver\win7 folder, and the driver for Windows 10 is in the \FPGA\USBC_FX3\demo_batch\Driver\win10 folder.
 - Use JP6, JP5, JP4 to set the PMODE[2:0] as “0F1” (F indicates floating).
 - Press KEY0 (RESET FX3).
 - Re-plug the Type-C cable one time, Cypress Control Center will show Cypress FX3 USB Streamer Example Device and BOS (SuperSpeed Device capability), it indicates the USB3.0 Has completed the setup. When LED1 lights up, it indicates USB signal is detected. LED0 lights off indicates Type-C Cable two sides plugged with same directions, LED0 lights up indicates Type-C Cable two sides plugged with reverse directions.
 - Execute the .exe application: \demo_batch\Host_app\loopback\bulkloop.exe, see **Figure 2-18** 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.

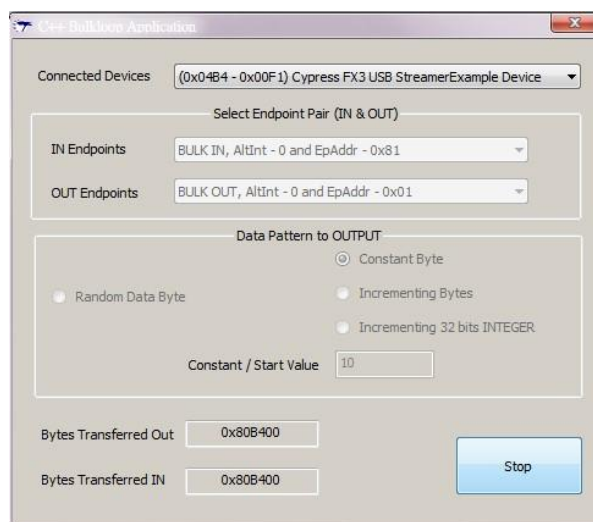


Figure 2-18 FX3 bulk in out bulkloop

Table 2-3 summarizes the functional keys and details of each LED status.

Table 2-3 The functional keys of the HAN Pilot Platform USBC_FX3 demonstration

Name	Description
LED0	LED0 lights off indicates Type-C Cable two sides plugged with same directions. LED0 lights up indicates Type-C Cable two sides plugged with reverse directions.
LED1	It lights up when the USB signal is detected.

- Program the FX3 firmware:(Optional)
 - Execute the demo batch file “test.bat” from the directory \FPGA\USBC_FX3\demo_batch
 - Execute Cypress Control:
FPGA\USBC_FX3\demo_batch\Host_app\ download_firmware\CyControl.exe
 - Use JP6, JP5, JP4 to set the PMODE[2:0] as “F11”(F indicates floating).
 - Press KEY0 (RESET FX3), Cypress Control center will show Cypress FX3 USB Bootloader Device.
 - Program software, In Control Center. Click Program FX3 SPI FLASH, as shown in **Figure 2-19**. Select file FPGA\USBC_FX3\demo_batch\FX3_Firmware\SF_loopback.img. Wait until It reports “Programming of SPI FLASH Succeeded”.

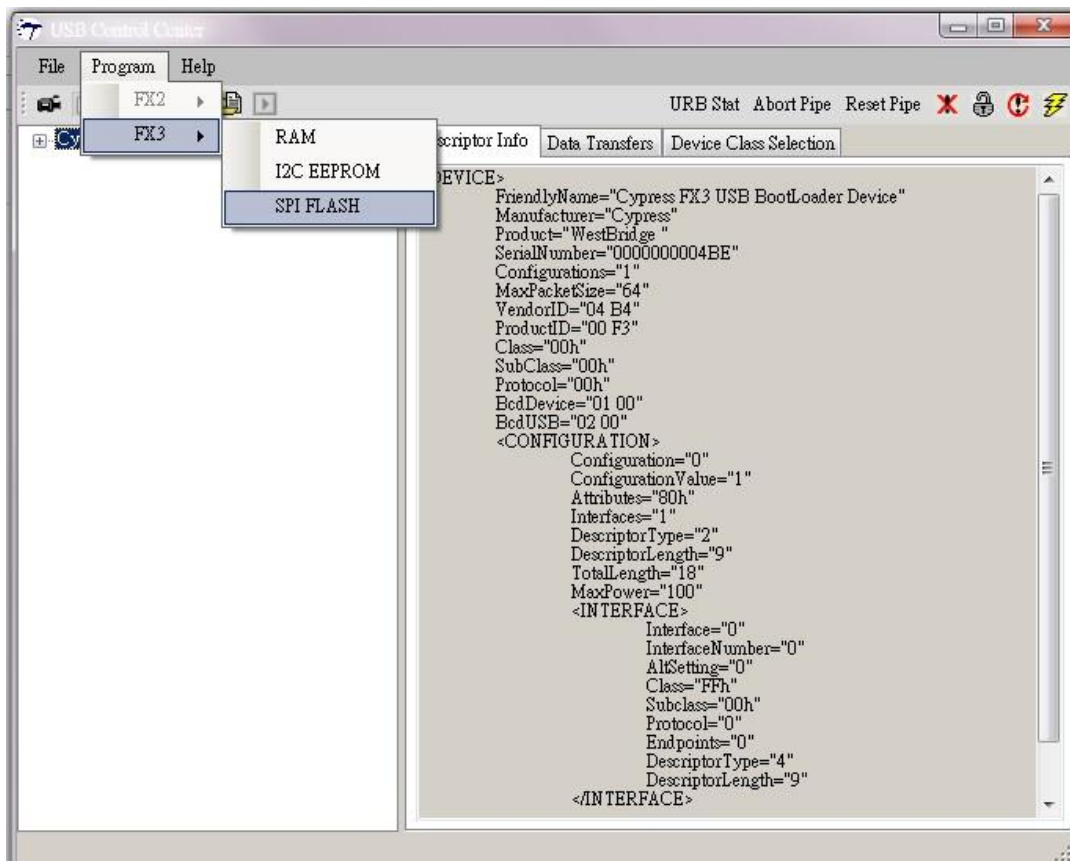


Figure 2-19 Cypress Control Center

2.7 HDMI TX and RX in 4K Resolution

This demonstration uses the Intel® FPGA HDMI IP core to implement the HDMI Retransmit function in the FPGA of the HAN Pilot Platform. As shown in **Figure 2-20**, user can connect an HDMI video player to the input video and audio data to HDMI RX port of the HAN Pilot Platform. After the HDMI video data is received in the FPGA, it will be instantly transferred to the HDMI TX port. The user only needs to connect an HDMI screen to the HAN Pilot Platform. User only needs to connect an HDMI screen to the HAN Pilot Platform, then you can watch the images output by the HDMI Player. This demonstration supports image resolution up to 4K60P. If you want to learn HDMI high-performance related image processing, this demo can help you learn quickly.

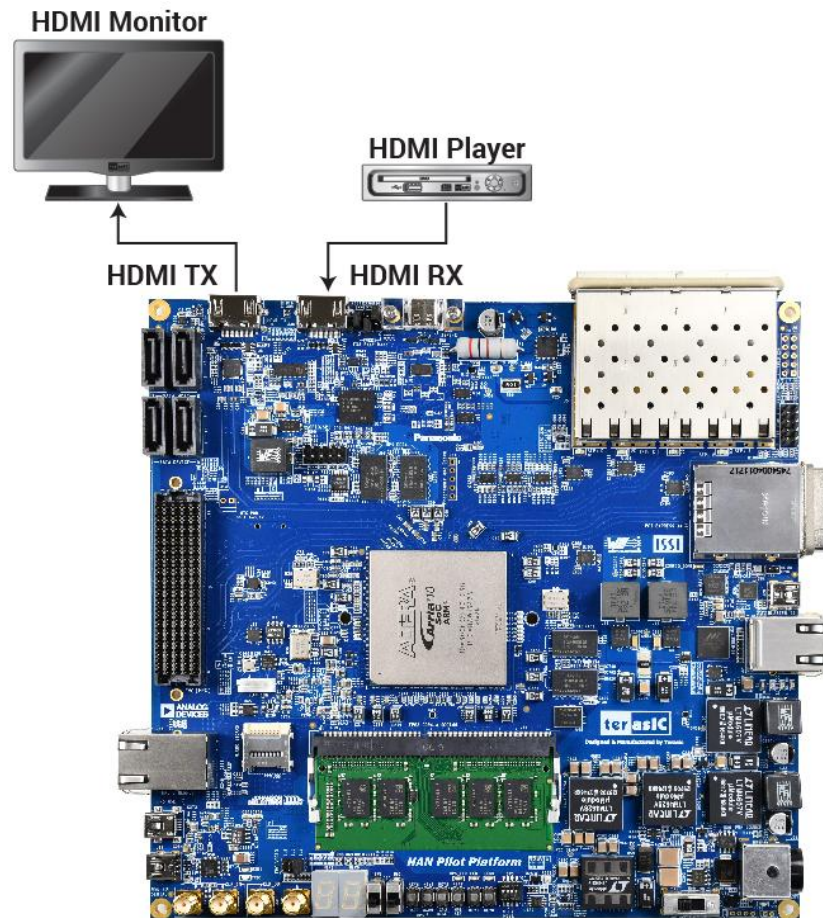


Figure 2-20 The Architecture of the demonstration

■ System Block Diagram

Figure 2-21 shows the system block diagram of this example. It shows that the Intel® FPGA HDMI IP core is used in the FPGA. It is divided into two parts: Transmitter and Receiver. These two IPs can be directly used by Transition-minimized differential signaling (TMDS) interface connection, only need HDMI repeater or redrive IC as an intermediary to connect HDMI devices, no need to use special HDMI Transmitter and Receiver IC. For details about HDMI IP, please refer to [HDMI Intel® FPGA IP User Guide](#) and [Intel FPGA HDMI Design Example User Guide for Intel Arria 10 Devices](#).

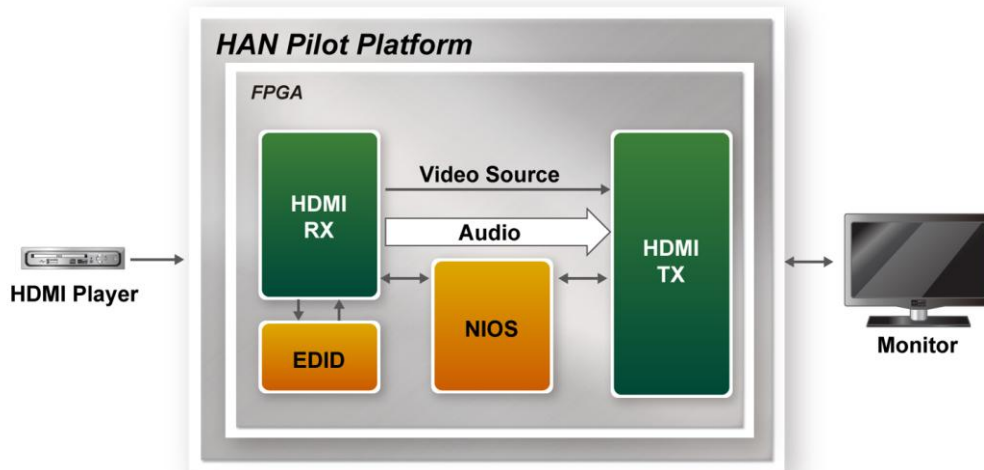


Figure 2-21 The system block of the demonstration

This demonstration first needs to connect the HAN Pilot Platform to the HDMI interface screen. Because this demo can support various screen resolutions. When the HAN Pilot Platform connect to the monitor, the HDMI RX block in the FPGA will first read the supporting resolution of the monitor from the it's EDID and stored in the EDID RAM in the HDMI RX block. The NIOS handles the control signals between the EDID and the HDMI IP in this demo.

The HDMI Video player is then connected to the HDMI RX port of the HAN Pilot Platform. When the RX instance in the FPGA receives a video source from the external video generator, the video and audio data then go through a loopback FIFO before it is transmitted to the TX instance. The final image and sound data will be displayed on the monitor connected to the TX end. For a more detailed IP block diagram in the example, refer to [Figure 2-22](#).

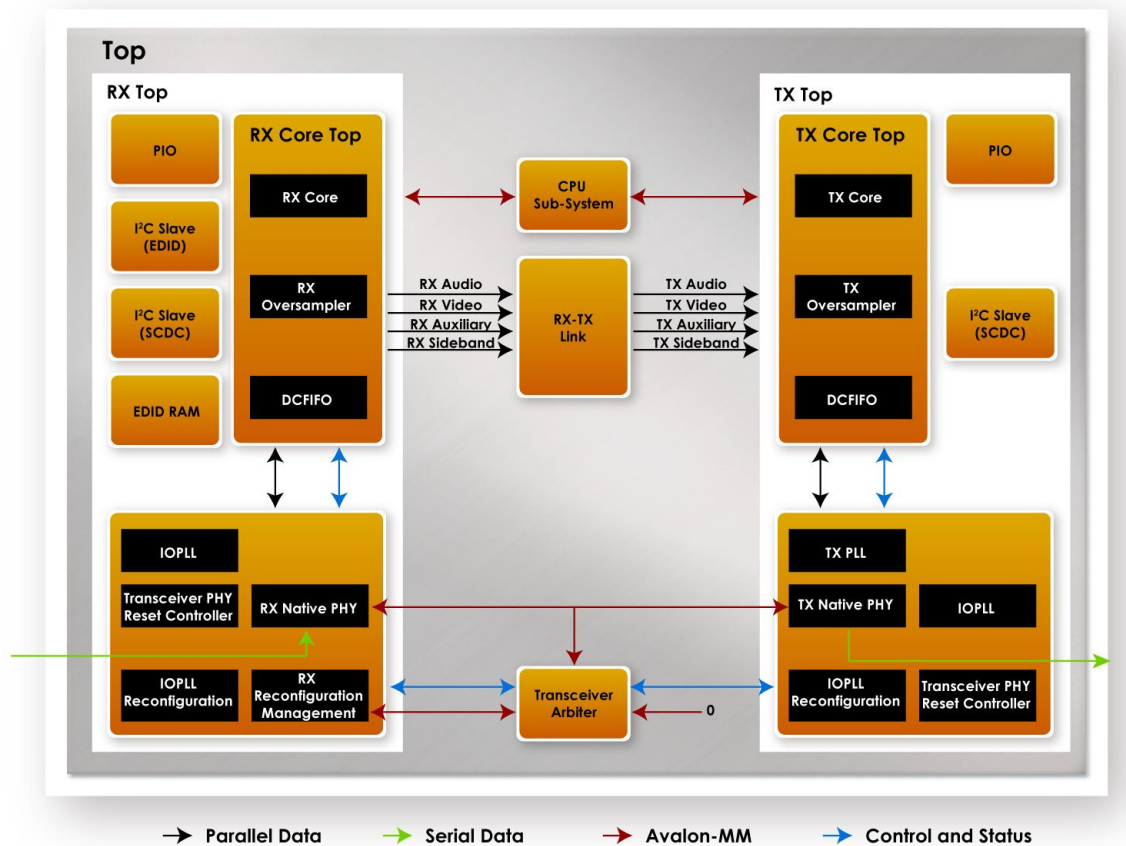


Figure 2-22 The HDMI IP block of the demonstration

■ Demonstration Source Code

Project Location: System CD\Demonstration\FPGA\HDMI_RX_TX\

- Quartus Project directory: HDMI_RX_TX\quartus\
- Nios II Eclipse: HDMI_RX_TX\software\
- Demonstration Batch File:

Demo Batch File Folder: HDMI_RX_TX\demo_batch\

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat, test.sh
- FPGA Configure File: a10_hdmi2_demo.sof
- Nios II Program: HDMI_RX_TX.elf

■ Hardware Requirement

- A PC
- An HDMI monitor capable of displaying 4K/60P
- An HDMI video player capable of outputting 4K/60P resolution

■ Demonstration Setup

1. Make sure Quartus Prime and Nios II are installed on your PC.
2. Set MSEL[2:0] to 010, Set SW0 to 1, SW1 to 0.
3. Connect a HDMI monitor to the HAN Pilot Platform as shown in **Figure 2-23**.

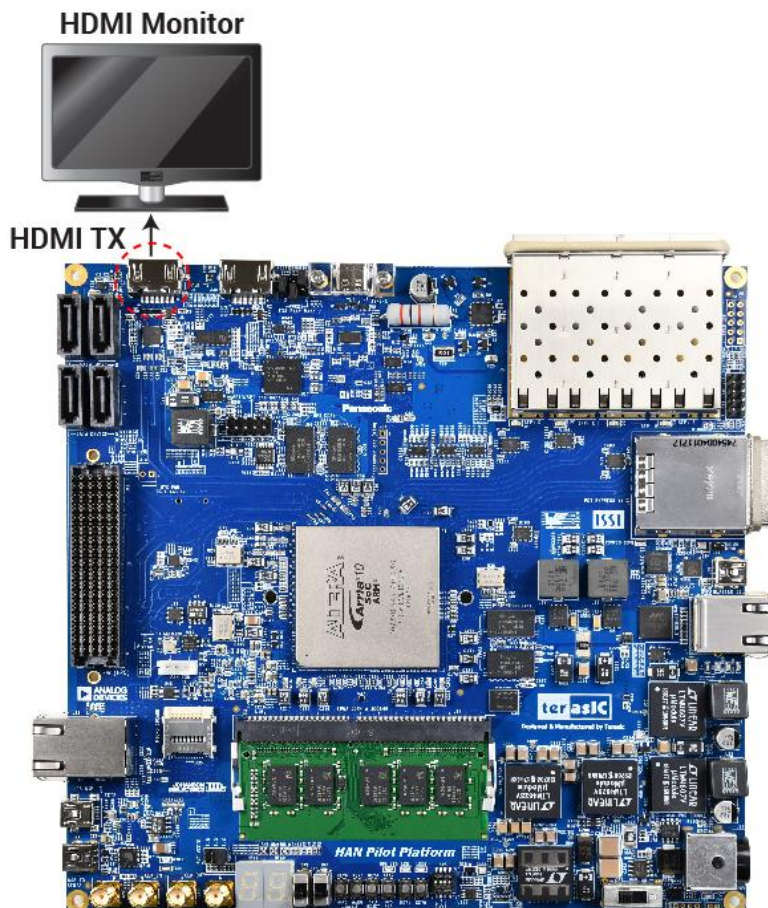
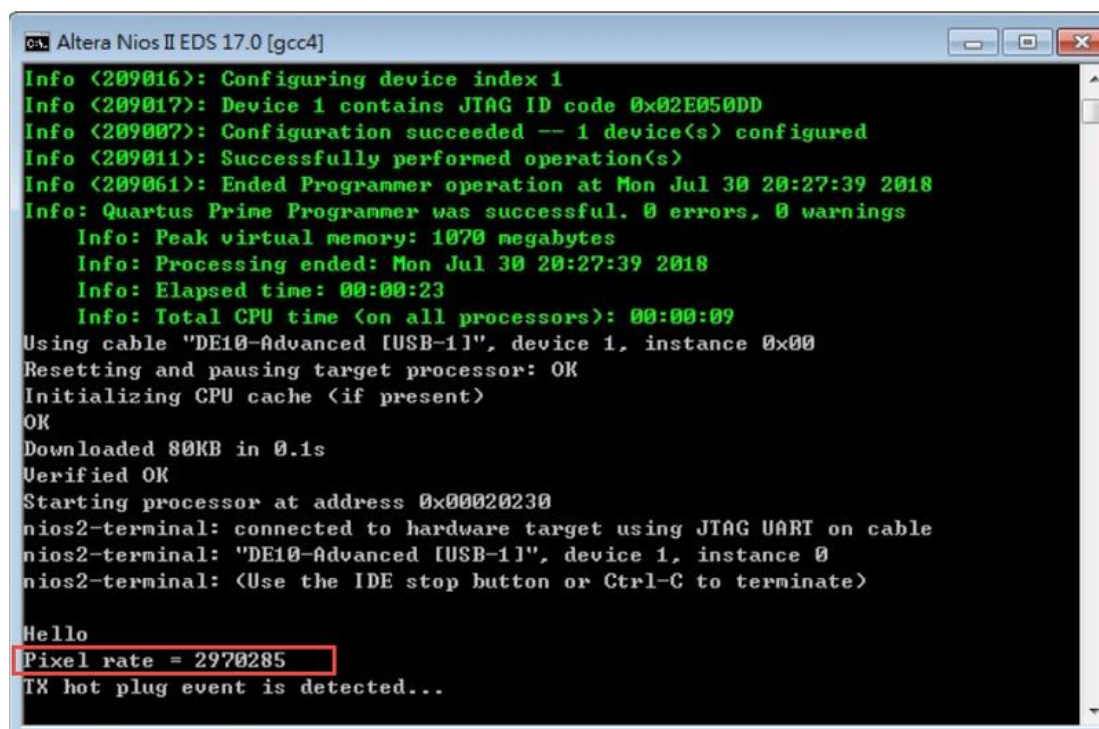


Figure 2-23 Connection setup of the HDMI TX monitor and HAN Pilot Platform

4. Connect the USB Blaster II port on the HAN Pilot Platform to the computer using the USB cable (Do not connect the HDMI video player at this time).
5. Open HAN Pilot Platform power and execute demo batch file: test.bat
6. Waiting for FPGA code download completed.
7. First set the output resolution of the HDMI video player to 4K@60Hz and connect to the HDMI RX port of HAN Pilot Platform as shown in **Figure 2-20**.
8. Observe the Nios command shell window (See **Figure 2-24**), whether the Pixel rate is around 297000 (the value when the resolution is 4K).
9. Check if the resolution of the HDMI monitor is 4K/60P. If the monitor cannot support 4K resolution, some HDMI video players will automatically switch to full HD or lower resolution. If the HDMI video player cannot automatically switch, please manually switch to the resolution that monitor can support.



```
Altera Nios II EDS 17.0 [gcc4]
Info <209016>: Configuring device index 1
Info <209017>: Device 1 contains JTAG ID code 0x02E050DD
Info <209007>: Configuration succeeded -- 1 device(s) configured
Info <209011>: Successfully performed operation(s)
Info <209061>: Ended Programmer operation at Mon Jul 30 20:27:39 2018
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1070 megabytes
Info: Processing ended: Mon Jul 30 20:27:39 2018
Info: Elapsed time: 00:00:23
Info: Total CPU time (on all processors): 00:00:09
Using cable "DE10-Advanced [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 80KB in 0.1s
Verified OK
Starting processor at address 0x00020230
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE10-Advanced [USB-1]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

Hello
Pixel rate = 2970285
TX hot plug event is detected...
```

Figure 2-24 Verify the pixel rate of the demonstration

2.8 HDMI TX in 4K Resolution

Compared with the section 2.7 HDMI TX and RX in 4K Resolution, the difference of this demonstration is that only the TX of the Intel® FPGA HDMI IP cores is used in the FPGA. This demo also has a video test pattern generator built into the FPGA. The highest resolution 4K image is sent to the HDMI TX IP. It is displayed via an external HDMI monitor.

■ System Block Diagram

Figure 2-25 shows the system block diagram of the demo. First, Nios is used to generate the test pattern output to the HDMI TX IP. The resolution of the generated pattern can be 4K or Full HD (1080). User can switch the output resolution instantly through the Switch on the HAN Pilot Platform. The HDMI TX IP is identical to section 2.7.

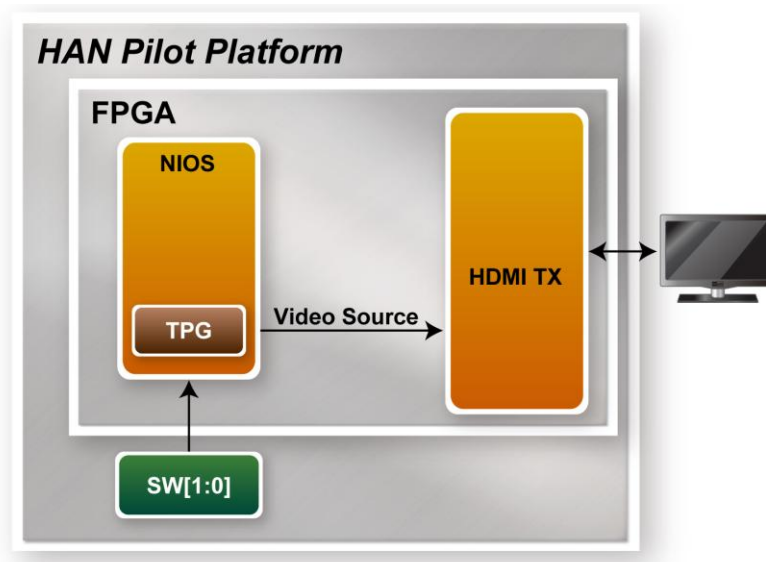


Figure 2-25 The System Block Diagram of the demonstration

■ Design Tools

- Quartus Prime 18.0.0 Standard Edition
- Nios II Software Build Tools for Eclipse 18.0

■ Demonstration Source Code

Project Location: System CD\Demonstration\FPGA\HDMI_TX_4K\

- Quartus Project directory: HDMI_TX_4K
- Nios II Eclipse: HDMI_TX_4K\software\
- Demonstration Batch File:

Demo Batch File Folder: HDMI_TX_4K\demo_batch\

The demo batch file includes following files:

- Batch File for USB-Blaster II: test.bat, test.sh
- FPGA Configure File: HDMI_TX_4K.sof
- Nios II Program: vip_control.elf

■ Hardware Requirement

- A PC
- An HDMI monitor capable of displaying 4K/60P

■ Demonstration Setup

1. Make sure Quartus Prime and Nios II are installed on your PC.
2. Connect a HDMI monitor to the HAN Pilot Platform as shown in **Figure 2-23**.
3. Connect the USB Blaster II port on the HAN Pilot Platform to the computer using the USB cable (do not connect the HDMI video player at this time).
4. Set MSEL[2:0] to 010.
5. Open HAN Pilot Platform power and execute demo batch file: test.bat

6. Waiting for FPGA code download completed
7. Check if the test color pattern is shown on the HDMI monitor (See **Figure 2-26**).

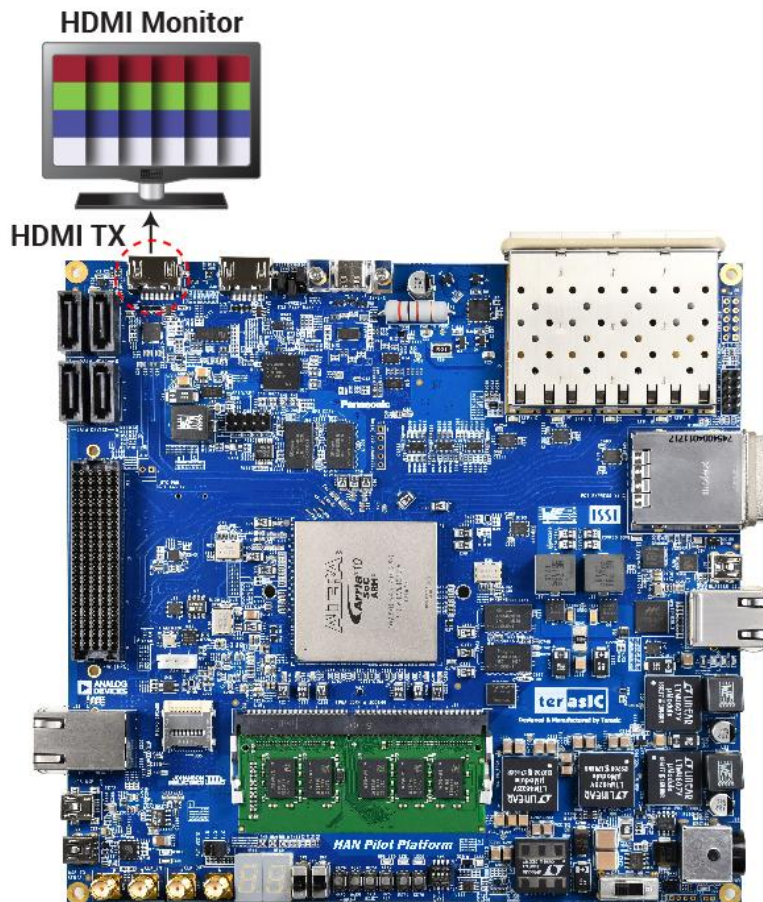


Figure 2-26 Test color pattern is shown on the HDMI monitor

8. User can switch SW[1:0] to change the resolution of the test patter output. The relationship between the detailed screen resolution and the switch is shown in **Table 2-4**.

Table 2-4 Switch setting for the resolution of the test pattern

SW[1:0]	Resolution Setting
00	1080@60P
01	4K@60P
10	1080@60P (Same with the SW[1:0] = 00)
11	4K@30P

2.9 Low Latency Ethernet 10G MAC Demo

This 10GBASE-R Ethernet design example is generated according to the documents : [Low Latency Ethernet 10G MAC Intel Arria 10 FPGA IP Design Example User Guide](#). The LL (Low Latency) 10GbE IP is used in the example design. This example executes the external loopback test through

one of the SFP+ ports on the FPGA main board. A SFP+ loopback fixture is required to perform this demonstration. **Figure 2-27** shows the block diagram of this demonstration.

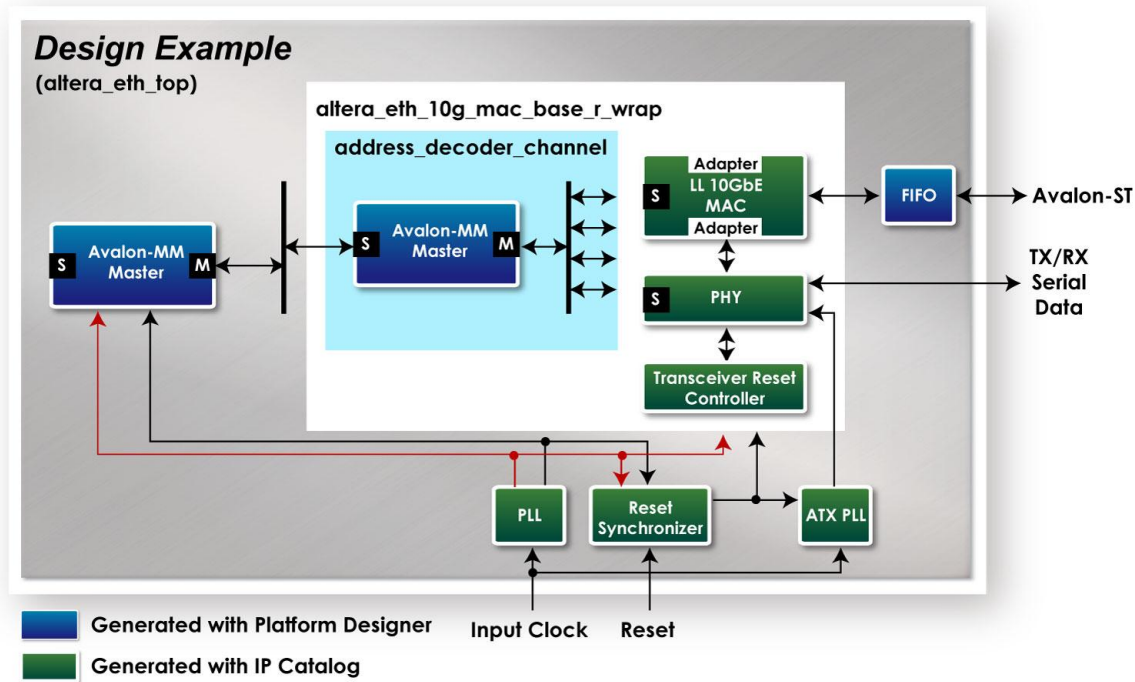


Figure 2-27 Block diagram of 10GBASE-R demo

■ Project Information

The Project information is shown in the **Table 2-5**.

Table 2-5 Project Information

Item	Description
Project Location	CDROM/Demonstration/FPGA/alt_eth
FPGA Bit Stream	CDROM/Demonstration/FPGA/alt_eth/output_files/altera_eth_top.sof
Test Scrip File	CDROM/Demonstration/FPGA/alt_eth/hwtesting/system_console/ gen_conf.tcl monitor_conf.tcl show_stats.tcl
Quartus Version	Quartus Prime 18.0.0 Build 614 Standard Edition

■ Demonstration Setup

Here is the procedure to setup the demonstration. A SFP+ loopback fixture is required for this

demonstration.

1. Insert a SFP+ loopback fixture into the SFP+-A port on the HAN Pilot Platform as shown in **Figure 2-28**.

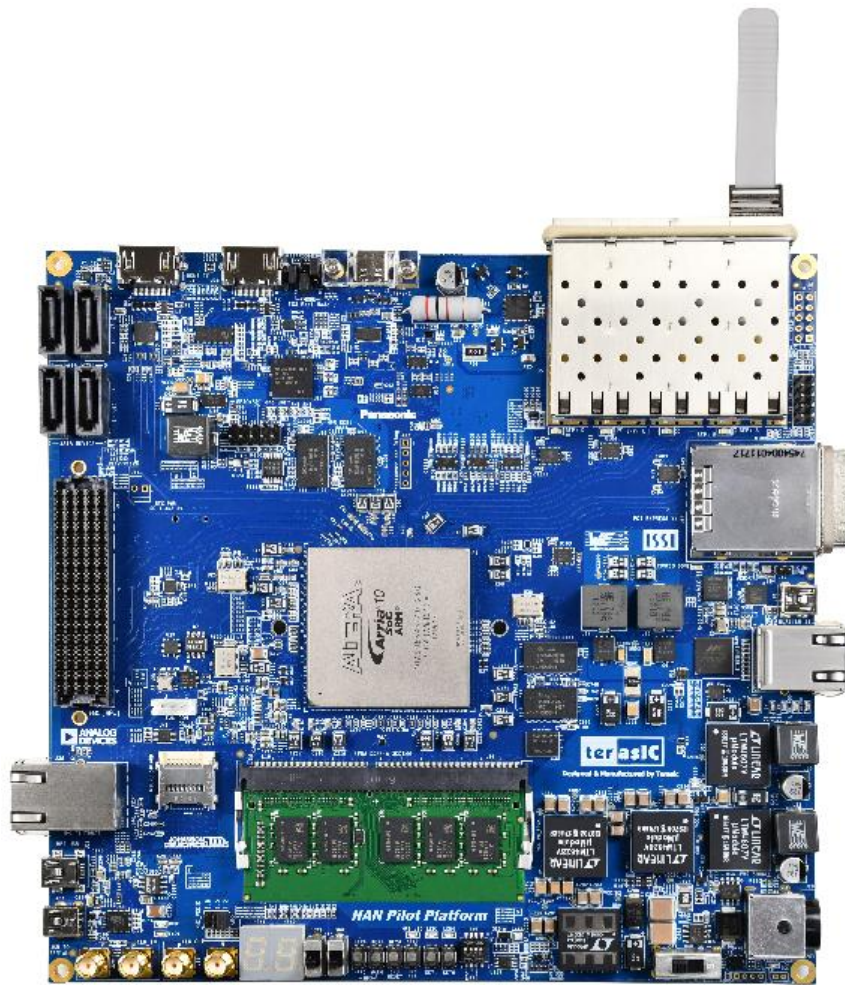


Figure 2-28 SFP+-A loopback on the HAN Pilot Platform

2. Connect the host PC to the FPGA board using a Mini-USB cable. Please make sure the USB-Blaster II driver is installed on the host PC.
3. Set MSEL[2:0] to 010.
4. Power on the FPGA Board.
5. Using Quartus to open the quartus project “altera_eth_top.qpf”.
6. Execute the demo batch file “test.bat” under the batch file folder, alt_eth\demo_batch.
7. Launch the System Console by selecting the menu item “Tools=>System Debugging Tools=>System Console” in Quartus.
8. In the Tcl Console pane, type “cd hwtesting/system_console” to change directory to the folder: ./alt_eth/hwtesting/system_console as shown in **Figure 2-29**.
9. Type “source gen_conf.tcl” to generates and sends about 4 billion packets as shown in **Figure 2-30**.
10. Type “source monitor_conf.tcl” to checks the number of good and bad packets received. as

shown in [Figure 2-31](#).

11. Type “source show_stats.tcl” to check the number of good and bad packets received. as shown in [Figure 2-32](#).
12. Wait 6 minutes to complete loopback task, then re-type “source monitor_conf.tcl” to see “0xffff2000” good packets.

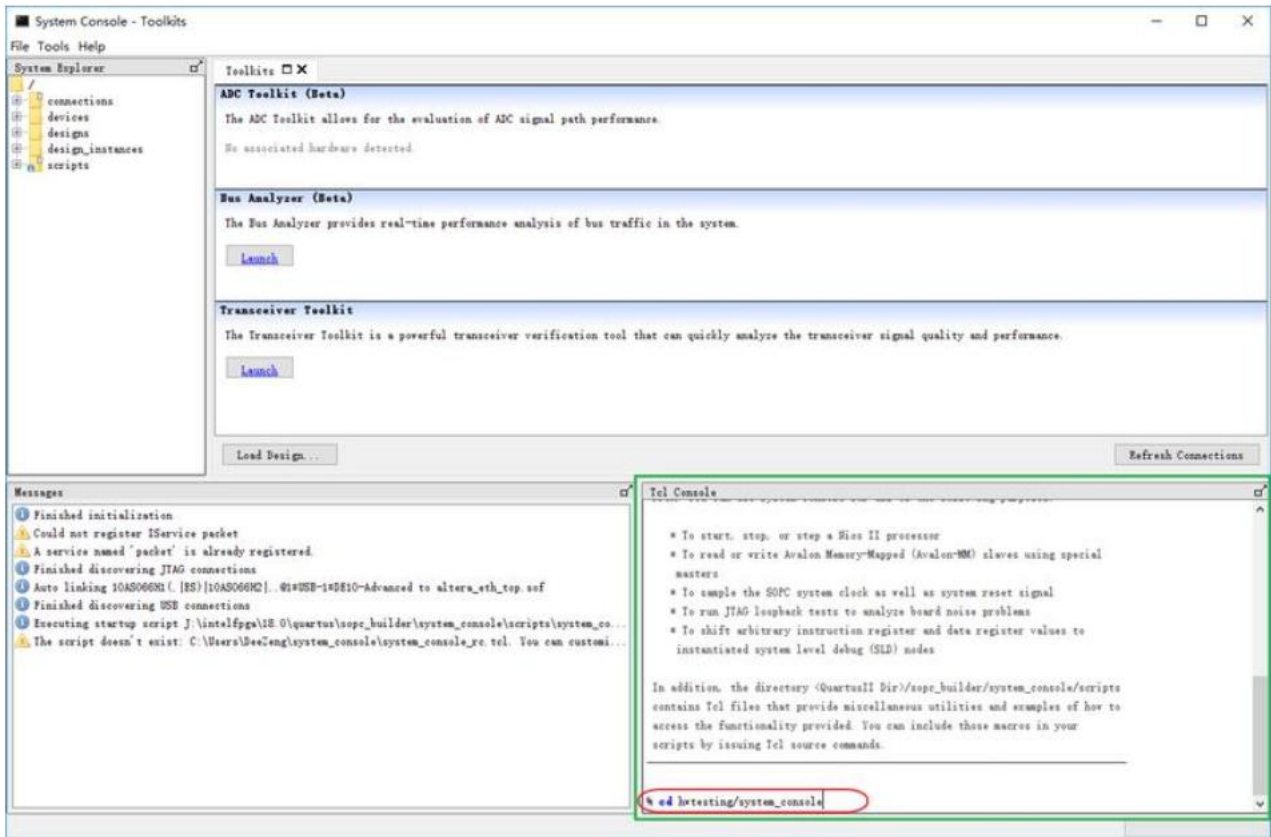


Figure 2-29 Launch the System Console for Ethernet 10GBASE-R Demo

```

Tcl Console
% cd hwtesting/system_console

% source gen_conf.tcl

Info: Opened JTAG Master Service

Read CRC remove status

CRC remove status before write:0x00000001

CRC remove status after write:0x00000000

Configure generator

Generator packet length before write:0x00000000

Generator packet length after write:0x00000050

Generator payload status before write:0x00000000

Generator payload status after write:0x00000001

Generator number of packets before write:0x00000000

Generator number of packets after write:0xffff2000

Generator start send packet before write:0x00000000

Generator after send packets after write:0x00000000

Info: Closed JTAG Master Service

% |

```

Figure 2-30 Ethernet 10BASE-R test message for gen_conf.tcl

```

% source monitor_conf.tcl

Info: Opened JTAG Master Service

Configure monitor

Check number of good packet:0x2302edd5

Check number of bad packet:0x00000000

Info: Closed JTAG Master Service

```

Figure 2-31 Ethernet 10BASE-R test message for monitor_conf.tcl

```

% source show_stats.tcl

Info: Opened JTAG Master Service

-----
Accessing Ethernet 10G MAC CSR
-----

-----
Reading TX Statistics
-----

clr : 0x00000000

framesOK : 0x00000000305b456a

framesErr : 0x0000000000000000

framesCRCErr : 0x0000000000000000

octetsOK : 0x0000000bb6acaece

pauseMACCtrlFrames : 0x0000000000000000

ifErrors : 0x0000000000000000

unicastFramesOK : 0x00000000305fd19d

unicastFramesErr : 0x0000000000000000

multicastFramesOK : 0x0000000000000000

```

Figure 2-32 Ethernet 10BASE-R test message for show_stats.tcl

2.10 Socket Server

The Arria 10 device on the HAN Pilot Platform consists of built-in serializer/reserialize (SERDES) circuitry for high-speed LVDS interfaces to support Gigabit Ethernet. Ethernet has been the dominant networking protocol providing a simple, cost-effective option for backbone and server connectivity. Gigabit Ethernet builds on top of the Ethernet protocol with speed up to 1000 Mbps, or 1 gigabit per second (Gbps). In this demonstration, we will illustrate how to create a simple socket server generated in Nios II using the Gigabit Ethernet devices equipped on the HAN Pilot Platform.

■ System Block Diagram

As indicated in the block diagram in [Figure 2-33](#), the Nios II processor is used to communicate with the client via Marvell 88E1111 Ethernet Transceiver.

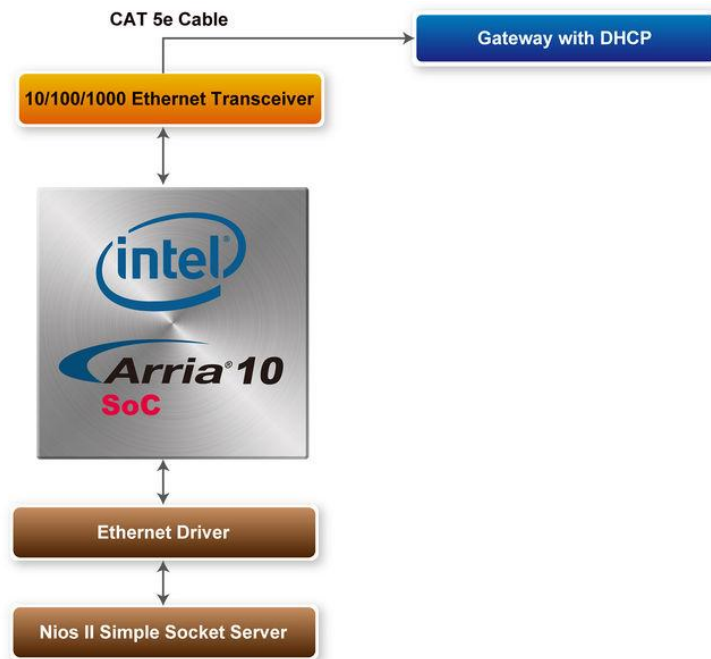


Figure 2-33 Block diagram of the Socket Server Demonstration

Part of Nios II, NicheStack TCP/IP Network Stack is a software suite of networking protocols designed to provide an optimal solution for designing network-connected embedded devices with the Nios II processor. A telnet client application is used to communicate with the Simple Socket Server issuing commands over a TCP/IP socket to the Ethernet-connected NicheStack TCP/IP Stack running on the HAN Pilot Platform with a Simple Socket Server. The Simple Socket Server continues to listen for commands on a TCP/IP port and operates the HAN Pilot Platform according to the commands from the telnet client. NicheStack TCP/IP stack uses the MicroC/OS-II RTOS multithreaded environment to provide immediate access to a stack for Ethernet connectivity for the Nios II processor. The Nios II processor system contains an Ethernet interface, or media access control (MAC)

■ How the Ethernet demonstration is built

In this following section we describe how to build the demonstration through the QSYS. The QSYS system includes the CPU processor, On-Chip memory, JTAG UART, system ID, timer, Triple-Speed Ethernet, Scatter-Gather DMA Controllers and peripherals which are linked together contained in the Nios II hardware system that are used when building a project. **Figure 2-34** presents the overall setup of the QSYS from the Ethernet Simple Socket Server project.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> ext_clk_50	Clock Source		exported				
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> iopl0_0	IOPLL Intel FPGA IP		ext_clk_50				
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> nios2_gen2	Nios II Processor						
		clk	Clock Input	Double-click to export	iopl0_0...				
		reset	Reset Input	Double-click to export	[clk]				
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		irq	Interrupt Receiver	Double-click to export	[clk]			IRQ 0	IRQ 31
		debug_reset_request	Reset Output	Double-click to export	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0140_0800	0x0140_0fff		
		custom_instruction_ma...	Custom Instruction Master	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> onchip_memory2_0	On-Chip Memory (RAM or ROM) Intel...		ext_clk_50	0x0180_0000	0x01ad_c6bf		
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> tse_mac	Triple-Speed Ethernet Intel FPGA IP						
		control_port_clock_co...	Clock Input	Double-click to export	iopl0_0...				
		reset_connection	Reset Input	Double-click to export	[control_port...				
		control_port	Avalon Memory Mapped Slave	Double-click to export	[control_port...	0x0140_1000	0x0140_13ff		
		receive_clock_connect...	Clock Input	Double-click to export	iopl0_0...				
		transmit_clock_conec...	Clock Input	Double-click to export	iopl0_0...				
		receive	Avalon Streaming Source	Double-click to export	iopl0_0...				
		transmit	Avalon Streaming Sink	Double-click to export	[transmit_clo...				
		mac_mdio_connection	Conduit	tse_mac_mac_mdio_con...					
		mac_misc_connection	Conduit	tse_mac_mac_misc_con...					
		pcs_ref_clk_clock_con...	Clock Input	tse_mac_pcs_ref_clk.cl...	exported				
		serial_connection	Conduit	tse_mac_serial_connecti...					
		status_led_connection	Conduit	tse_mac_status_led_con...					
		serdes_control_connect...	Conduit	tse_mac_serdes_control...					
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> msgdma_tx	Modular Scatter-Gather DMA Intel FPG...						
		mm_read	Avalon Memory Mapped Master	Double-click to export	[clock]				
		descriptor_read_master	Avalon Memory Mapped Master	Double-click to export	[clock]				
		descriptor_write_master	Avalon Memory Mapped Master	Double-click to export	[clock]				
		clock	Clock Input	Double-click to export	iopl0_0...				
		reset_n	Reset Input	Double-click to export	[clock]				
		csr	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0140_1640	0x0140_165f		
		prefetcher_csr	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0140_1600	0x0140_161f		
		csr_irq	Interrupt Sender	Double-click to export	[clock]				
		st_source	Avalon Streaming Source	Double-click to export	[clock]				
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> msgdma_rx	Modular Scatter-Gather DMA Intel FPG...		iopl0_0...	multiple		multiple	
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/> descriptor_memory	On-Chip Memory (RAM or ROM) Intel...		iopl0_0...	0x0140_2000	0x0140_3fff		

Figure 2-34 QSYS for Ethernet Simple Socket Server

In the Triple-Speed Ethernet IP Core configuration, the interface is set to SGMII as well as using the internal FIFO shown in **Figure 2-35**.

Parameters
System: System Path: tse_mac
Triple-Speed Ethernet Intel FPGA IP
altera_eth_tse
Details
Generate Example Design...
Core Configurations MAC Options FIFO Options Timestamp Options PCS/Transceiver Options
Core Variations
Core variation: 10/100/1000Mb Ethernet MAC with 1000BASE-X/SGMII PCS
Enable ECC protection
10/100/1000Mb Ethernet MAC
Interface: MII/SGMII
Use clock enable for MAC
Use internal FIFO
Number of ports: 1
1000BASE-X/SGMII PCS
Transceiver type: LVDS I/O

Figure 2-35 Triple-Speed Ethernet core configurations

In the MAC options section, the MDIO module is included that controls the PHY Management Module associated with the MAC block. The host clock divisor is to divide the MAC control register interface clock to produce the MDC clock output on the MDIO interface. The MAC control register interface clock frequency is 100 MHz and the desired MDC clock frequency is 2.5 MHz, a host clock divisor of 40 should be used. Once the Triple-Speed Ethernet IP configuration has been set and necessary hardware connections has been made click on ‘Generate’ to build the interconnect

logic automatically. In this following section we will describe the steps to create the Simple Socket Server using Nios II. We create a new project in Nios II using the project template, Simple Socket Server shown in **Figure 2-36**. The PTF file created using the SOPC builder in Quartus II is used in the Select Target Hardware section.

The screenshot shows the 'Nios II Project Wizard' dialog box. It is divided into three main sections: 'Target hardware information', 'Application project', and 'Project template'.
 - In 'Target hardware information', 'SOPC Information File name' is set to 'D:\my_file\SVN\de10_advanced\test_rev1' and 'CPU name' is set to 'nios2_gen2'.
 - In 'Application project', 'Project name' is 'Socket_Server'. The 'Use default location' checkbox is unchecked, and 'Project location' is 'file\SVN\de10_advanced\software\Socket_Server'.
 - In 'Project template', a list of templates is shown on the left, with 'Simple Socket Server' selected. On the right, a 'Template description' box explains that the Simple Socket Server uses the industry standard Sockets interface to TCP/IP and shows how to initialize the NicheStack TCP/IP Stack and run a simple TCP server application.

Figure 2-36 Nios II project simple socket server

■ Overview

The Simple Socket Server uses the industry standard sockets interface to TCP/IP. It uses DHCP protocol to requests a valid IP from the Gateway. During the device initialization process, the NicheStack TCP/IP Stack system code calls `get_mac_add()` and `get_ip_add()` to get the MAC and IP addresses for the network interface. Once the MAC address is generated, auto-negotiation is initiated where both connected devices, the Ethernet (Marvel 88E1111) and Gateway devices broadcasts its transmission parameters, speed and duplex mode. By default, the MAC interface for the Ethernet device is set to SGMII. In this demonstration, we are using SGMII MAC interface which can be configured through the management interface of the 88E1111 Ethernet device. Once the link is established an IP address is assigned to the Ethernet device along with the port number. Through TCP and port number, the demonstration uses Telnet client to establish connection with the Simple Socket Server, where it is continuously listening on the port. Once the connection is established between the Telnet client and Simple Socket Server, the Telnet client is able to send packets which are received by the Nios II processor and through the Simple Socket Server it will send server command to the HAN Pilot Platform. The packet sent contains LED command which is extracted and dispatched to the LED command queue for processing by the LED management tasks. **Figure 2-37** shows the software architecture of the Nios program for the Simple Socket Server. The top block containing the Nios II processor and the necessary hardware to be implemented into the HAN Pilot Platform. The software device drivers contain the necessary device drivers needed for Ethernet and other hardware components to functions. The HAL API block provides the interface

for the software device drivers, while the MicroC/OS-II provides communication services to the NicheStack and the Simple Socket Server. The NicheStack TCP/IP stack software block provides networking services to the application where it contains tasks for Simple Socket Server and LED management.

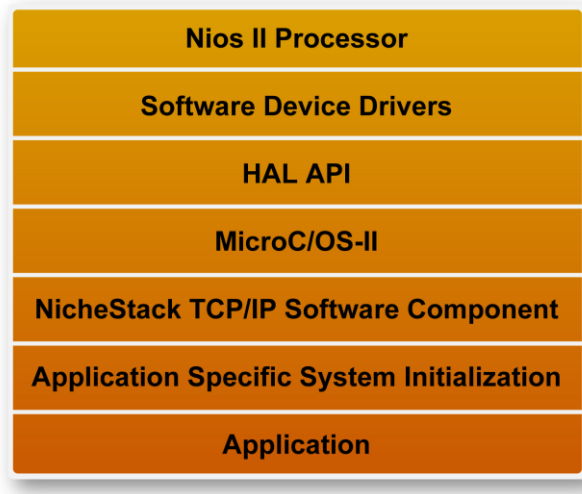


Figure 2-37 Nios program software architecture

■ Running the demonstration

- Design Tools
 - Quartus Prime 17.0 Standard Edition
 - Nios II Software Build Tools for Eclipse 17.0
- Demonstration Source Code
 - Quartus Project directory: Socket_Server
 - Nios II Eclipse: Socket_Server \software
- Nios II Project Compilation

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

- Demonstration Batch File

Demo Batch File Folder: Socket_Server \demo_batch

The demo batch file includes following files:

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

- Demonstration Setup

Please follow below procedures to setup the demonstration.

1. Make sure Quartus Prime and Nios II are installed on your PC.
2. Use USB Cable to connect PC and the FPGA board and install USB Blaster II driver if necessary.
3. Connect the ethernet to Router or network switch with DHCP.
4. Set MSEL[2:0] to 010.

5. Power on the FPGA board.
6. Execute the demo batch file “test.bat” under the folder “Socket_Server\demo_batch”.
7. After Nios II program is downloaded and executed successfully, a prompt message will be displayed in nios2-terminal. Where the IP address and port numbers are assigned as shown below in **Figure 2-38**.

```

Altera Nios II EDS 17.0 [gcc4]
Your Ethernet MAC address is 00:07:ed:12:8f:ff
prepped 1 interface, initializing...
[tse_mac_init]
INFO : TSE MAC 0 found at address 0x01403000
INFO : PHY Marvell 88E1111 found at PHY address 0x00 of MAC Group[0]
INFO : PHY[0.0] - Automatically mapped to tse_mac_device[0]
INFO : PHY[0.0] - Restart Auto-Negotiation, checking PHY link...
INFO : PHY[0.0] - Auto-Negotiation PASSED
INFO : PCS[0.0] - Configuring PCS operating mode
INFO : PCS[0.0] - PCS SGMII mode enabled
INFO : PHY[0.0] - Checking link...
INFO : PHY[0.0] - Link established
INFO : PHY[0.0] - Speed = 1000, Duplex = Full
OK, x=0, CMD_CONFIG=0x00000000

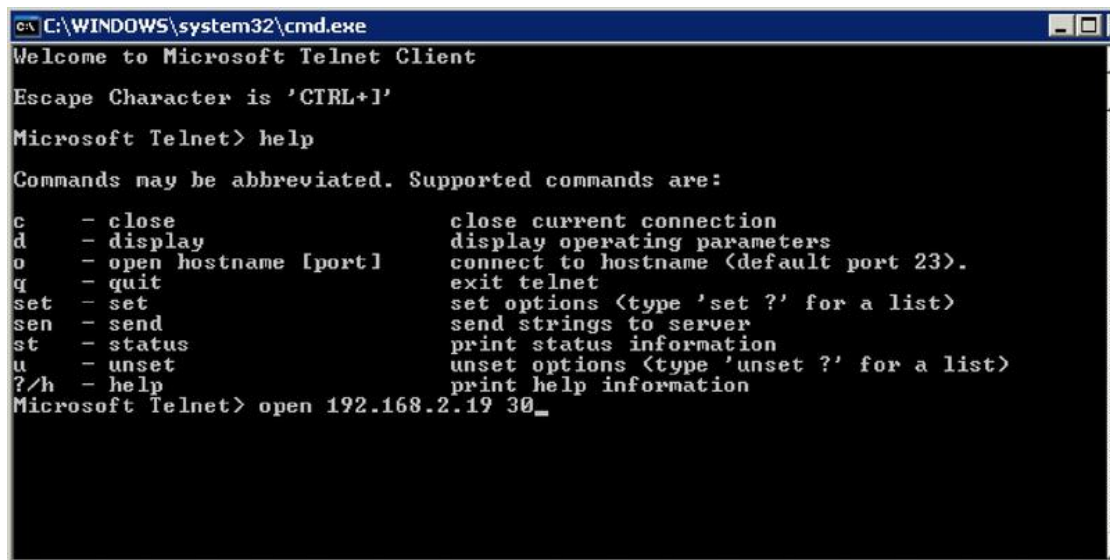
MAC post-initialization: CMD_CONFIG=0x0400020b
[tse_sgdma_read_init] RX descriptor chain desc (1 depth) created
mctest init called
IP address of et1 : 192.168.21.20
Created "Inet main" task (Prio: 2)
Created "clock tick" task (Prio: 3)
No free buffers for rx
No free buffers for rx
No free buffers for rx
No free buffers for rx
No free buffers for rx
Acquired IP address via DHCP client for interface: et1
IP address : 192.168.21.160
Subnet Mask: 255.255.255.0
Gateway : 192.168.21.1

Simple Socket Server starting up
[ssr_task] Simple Socket Server listening on port 30
Created "simple socket server" task (Prio: 4)

```

Figure 2-38 Simple socket server

8. To establish connection, start the telnet client session by executing open_telnet.bat file and include the IP address assigned by the DHCP server-provided IP along with the port number as shown below in **Figure 2-39**.



```
C:\WINDOWS\system32\cmd.exe
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+I'
Microsoft Telnet> help
Commands may be abbreviated. Supported commands are:
c      - close                close current connection
d      - display              display operating parameters
o      - open hostname [port] connect to hostname (default port 23).
q      - quit                 exit telnet
set    - set                  set options (type 'set ?' for a list)
sen    - send                 send strings to server
st     - status               print status information
u      - unset                unset options (type 'unset ?' for a list)
?/h   - help                  print help information
Microsoft Telnet> open 192.168.2.19 30_
```

Figure 2-39 Telnet Client

9. From the Simple Socket Server Menu, enter the commands in the telnet session. Entering a number from zero through one followed by a return causes the corresponding the LEDs (LED0-LED1) to toggle on or off on the HAN Pilot Platform.

2.11 Auto Fan Speed Control

This demonstration shows you how to adjust the fan rotation speed according to the FPGA chip temperature value. The fan rotation speed is adjusted according to the process shown in [Figure 2-40](#).

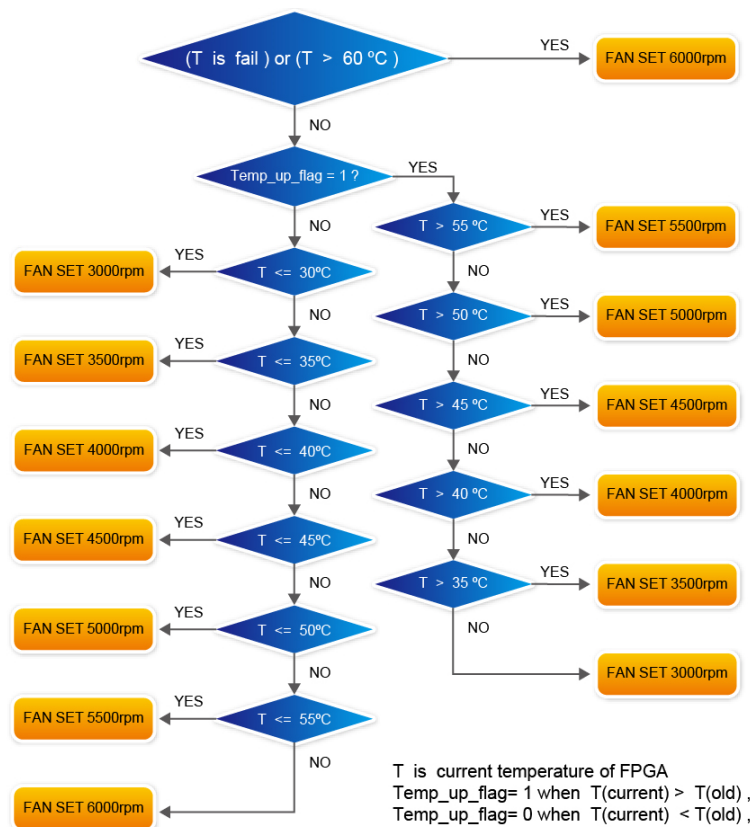


Figure 2-40 Fan rotation speed determination process

■ Function Block Diagram

Figure 2-41 Shows the Function block diagram of the AutoFan demonstration. This design has two main parts. The one is an I2C controller (FAN_TEMP_I2C), which can be used to read the temperature value of the HAN Pilot Platform Temperature Sensor IC (TMP441) and to set the Fan Controller IC (MAX6650) register for controlling the fan rotation speed. Both the Temperature Sensor and the Fan Controller use the same I2C bus. The Temperature Sensor I2C Slave-Address is 0x38, the Fan Controller I2C Slave-Address is 0x90. The other one is a judgment controller (FAN_ONOFF), which can determine the fan rotation speed value according to the process in **Figure 2-40**. In this demonstration, you can use the two seven segments to display the FPGA temperature value and the fan rotation speed value(rpm). All module functions are described as below:

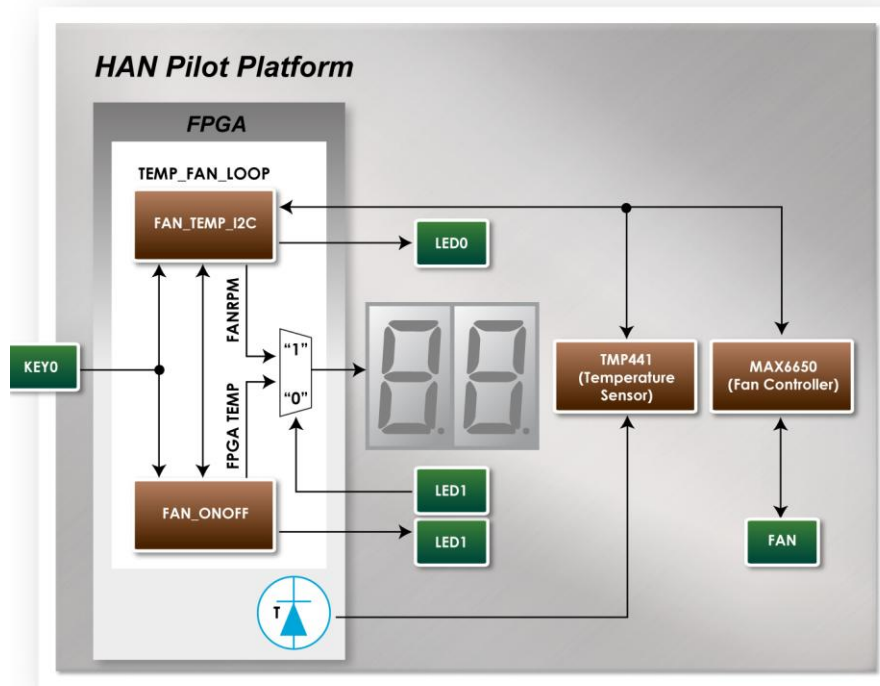


Figure 2-41 Block diagram of the AutoFan design

FAN_TEMP_I2C: This module can read the temperature value converted by the HAN Pilot Platform Temperature Sensor IC (TMP441) and to set the Fan Controller IC (MAX6650) register for controlling the fan rotation speed.

FAN_ONOFF: According to the FPGA temperature value (which output from FAN_TEMP_I2C module) and the process in [Figure 2-40](#), this module can determine the fan rotation speed value(rpm) and output it to the FAN_TEMP_I2C module.

KEY0: Which is used as system RESET function. When pressing KEY0, the two modules (FAN_TEMP_I2C and FAN_ONOFF) will reset.

SW1: Set the two seven segments to display the fan rotation speed or the FPGA temperature. SW1=1, display the current fan rotation speed value (display the thousands and the hundreds, in decimal), SW1=0, display FPGA temperature value.

The following are the descriptions of the platforms' set up, as well as the test steps.

■ AutoFan RTL Demonstration Setup

- Hardware Setting Up

As shown in [Figure 2-42](#):

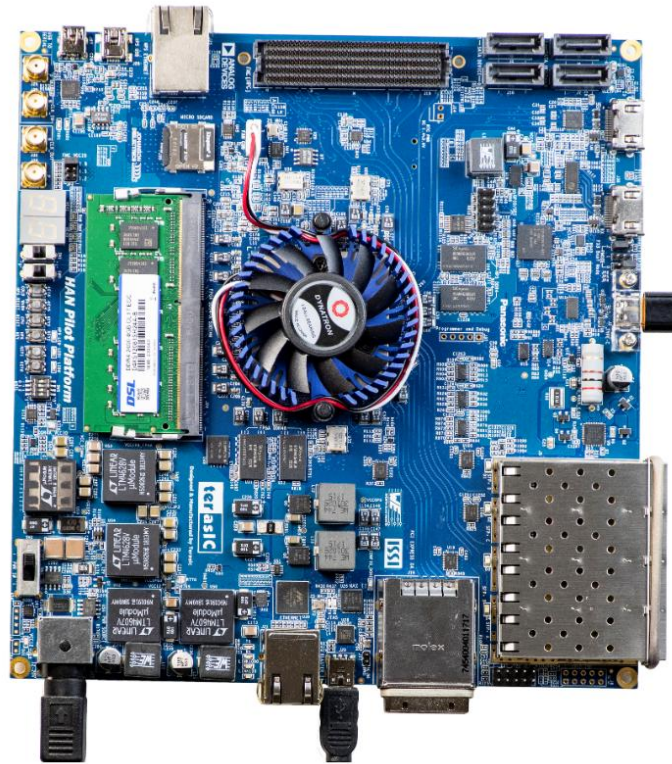


Figure 2-42 AutoFan demo hardware setting up

- Design Tools
 - Quartus Prime 18.0 Standard Edition
- Demonstration Source Code
 - Quartus project directory: AutoFan
 - Bitstream used: AutoFan.sof
- Demonstration Batch File
 - Demo batch file folder: demonstrations\AutoFan\demo_batch
- Demonstration Setup
 - Connect the HAN Pilot Platform USB Blaster II port (J20) to the host PC with a USB cable and install the USB-Blaster II driver if necessary.
 - Plug the 12V adapter to HAN Pilot Platform Board.
 - Set MSEL[2:0] to 010.
 - Power on the HAN Pilot Platform.
 - Execute the demo batch file “test.bat” from the directory \FPGA\AutoFan\demo_batch. The fan rotation speed value is finally stabilized at RPM=3000. set SW1=1, the two seven segments will display the current fan rotation speed value (display the thousands and the hundreds, in decimal), set SW1=0, the two seven segments will display the FPGA temperature value (display the tens and the ones, in decimal). LED[1:0] functions in this demonstration are: when the Fan rotation speed is abnormal (ex, fan doesn’t rotate). LED0 lights up, when the FPGA temperature value is greater than 50°C, LED1 lights up.

Table 2-6 Summarizes the functional keys and details of each LED status.

Table 2-6 The functional keys of the AutoFan demonstration

Name	Description
KEY0	System reset, press the KEY0 to reset the system
SW1	Control the two seven segments HEX[1:0] SW1=1, display the current fan rotation speed value(display the thousands and the hundreds, in decimal) SW1=0, display the FPGA temperature value(display the tens and the ones, in decimal)
HEX[1:0]	Display two decimal numbers
LED0	When the Fan rotation speed is abnormal (ex, fan doesn't rotate), LED0 lights up
LED1	When the FPGA temperature value is greater than 50℃, LED1 lights up

Chapter 3

Examples for HPS SoC

This chapter provides several C-code examples based on the Intel SoC Linux built by Yocto project. These examples demonstrate major features connected to HPS interface on HAN Pilot Platform such as users LED/KEY, Network Communication. All the associated files can be found in the directory Demonstrations/SOC of the HAN Pilot Platform System CD. Please refer to Chapter 5 "Running Linux on the HAN Pilot Platform" from the [HAN Pilot Platform Getting Start Guide](#).

To install the demonstrations on the host computer: Copy the directory Demonstrations into a local directory of your choice. Intel SoC EDS v18.0 is required for users to compile the c-code project.

3.1 User LED and KEY

■ Function Block Diagram

Figure 3-1 shows the function block diagram of this demonstration. The users LED and KEY are connected to the GPIO1 controller in HPS. The behavior of GPIO controller is controlled by the register in GPIO controller. The registers can be accessed by application software through the memory-mapped device driver, which is built into Intel SoC Linux.

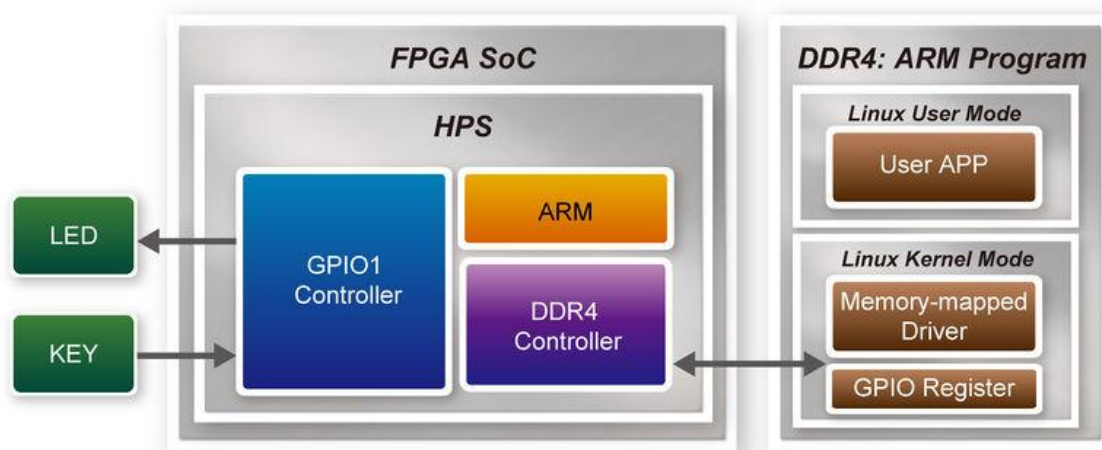


Figure 3-1 Block diagram of GPIO demonstration

■ Block Diagram of GPIO Interface

The HPS provides three general-purpose I/O (GPIO) interface modules. **Figure 3-2** shows the block diagram of GPIO Interface. GPIO[28..0] is controlled by the GPIO0 controller and GPIO[57..29] is controlled by the GPIO1 controller. GPIO[70..58] and input-only GPI[13..0] are controlled by the

GPIO2 controller.

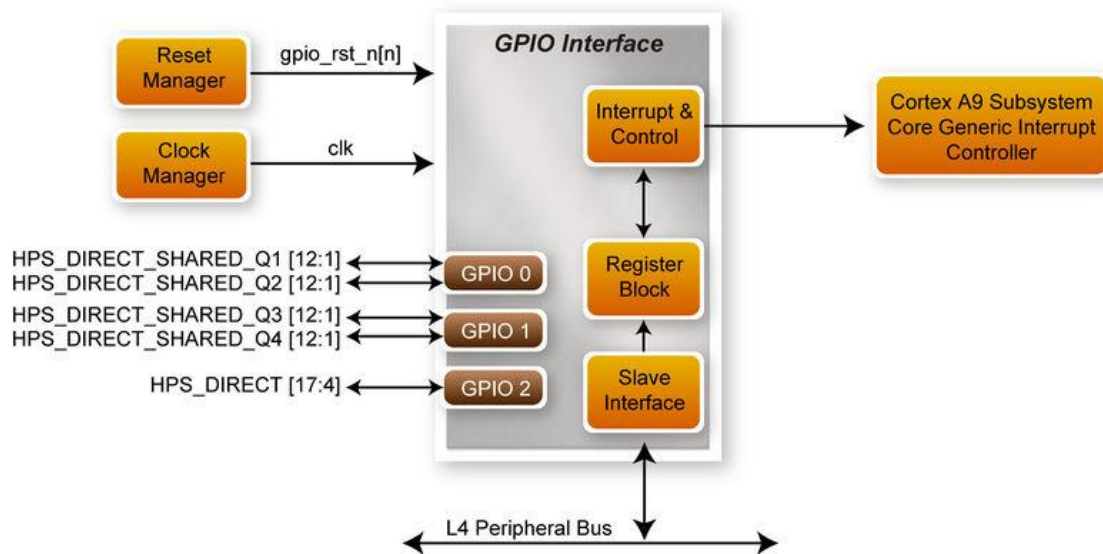


Figure 3-2 Block diagram of GPIO Interface

■ GPIO Register Block

The behavior of I/O pin is controlled by the registers in the register block. There are three 32-bit registers in the GPIO controller used in this demonstration. The registers are:

- **gpio_swporta_dr**: write output data to output I/O pin
- **gpio_swporta_dds**: configure the direction of I/O pin
- **gpio_ext_porta**: read input data of I/O input pin

The **gpio_swporta_dds** configures the LED pin as output pin and drives it high or low by writing data to the **gpio_swporta_dr** register. The first bit (least significant bit) of **gpio_swporta_dds** controls the direction of first IO pin in the associated GPIO controller and the second bit controls the direction of second IO pin in the associated GPIO controller and so on. The value "1" in the register bit indicates the I/O direction is output, and the value "0" in the register bit indicates the I/O direction is input.

The first bit of **gpio_swporta_dr** register controls the output value of first I/O pin in the associated GPIO controller, and the second bit controls the output value of second I/O pin in the associated GPIO controller and so on. The value "1" in the register bit indicates the output value is high, and the value "0" indicates the output value is low.

The status of KEY can be queried by reading the value of **gpio_ext_porta** register. The first bit represents the input status of first IO pin in the associated GPIO controller, and the second bit represents the input status of second IO pin in the associated GPIO controller and so on. The value "1" in the register bit indicates the input state is high, and the value "0" indicates the input state is low.

■ GPIO Register Address Mapping

The registers of HPS peripherals are mapped to HPS base address space 0xFC000000 with 64MB size. The registers of the GPIO1 controller are mapped to the base address 0xFFC02A00 with 256B size, and the registers of the GPIO2 controller are mapped to the base address 0xFFC02B00 with 256B size, as shown in **Figure 3-3**.

GPIO0	GPIO0 module	0xFFC02900	256 B
GPIO1	GPIO1 module	0xFFC02A00	256 B
GPIO2	GPIO2 module	0xFFC02B00	256 B

Figure 3-3 GPIO address map

■ Software API

Developers can use the following software API to access the register of GPIO controller.

- **open:** open memory mapped device driver
- **mmap:** map physical memory to user space
- **alt_read_word:** read a value from a specified register
- **alt_write_word:** write a value into a specified register
- **munmap:** clean up memory mapping
- **close:** close device driver

Developers can also use the following MACRO to access the register.

- **alt_setbits_word:** set specified bit value to one for a specified register
- **alt_clrbits_word:** set specified bit value to zero for a specified register

The program must include the following header files to use the above API to access the registers of GPIO controller.

```
include <stdio.h>
include <unistd.h>
include <fcntl.h>
include <sys/mman.h>
include "hwlib.h"
include "socal/socal.h"
include "socal/hps.h"
include "socal/alt_gpio.h"
```

■ LED and KEY Control

Figure 3-4 shows the HPS users LED and KEY pin assignment for the HAN Pilot Platform. The LED is connected to GPIO1_IO1 and the KEY is connected to GPIO1_IO4.

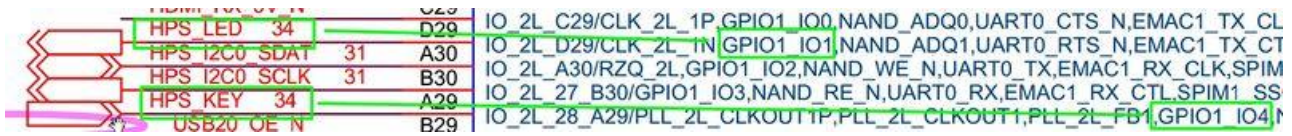


Figure 3-4 Pin assignment of LED and KEY

Figure 3-5 shows the **gpio_swporta_ddr** register of the GPIO1 controller. The bit-0 controls the pin direction of GPIO1_IO0. The bit-1 controls the pin direction of GPIO1_IO1, which connects to HPS_LED, the bit-4 controls the pin direction of GPIO1_IO4, which connects to HPS_KEY and so on. The pin direction of HPS_LED and HPS_KEY are controlled by the bit-1 and bit-4 in the **gpio_swporta_ddr** register of the GPIO1 controller, respectively. Similarly, the output status of HPS_LED is controlled by the bit-1 in the **gpio_swporta_dr** register of the GPIO1 controller. The status of KEY can be queried by reading the value of the bit-4 in the **gpio_ext_porta** register of the GPIO1 controller.

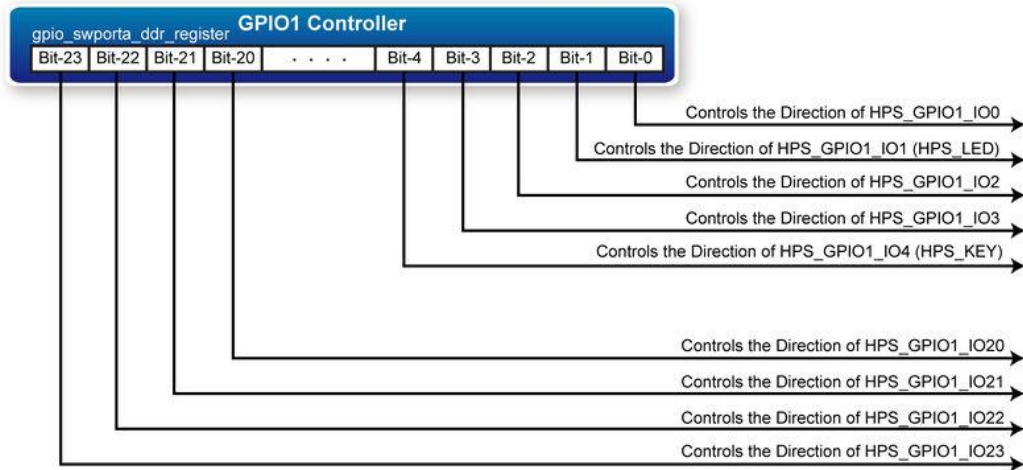


Figure 3-5 Gpio swporta ddr register in the GPIO1 controller

The following mask is defined in the demo code to control LED and KEY direction and LED's output value.

```
define USER_IO_DIR (0x00000002)
define BIT_LED (0x00000002)
define BUTTON_MASK (0x00000010)
```

The following statement is used to configure the LED associated pins as output pins.

```
alt_setbits_word( ( virtual_base + ALT_GPIO_SWPORTA_DDR_OFST ), USER_IO_DIR );
```

The following statement is used to turn on the LED.

```
alt_clrbits_word( ( virtual_base + ALT_GPIO_SWPORTA_DR_OFST ), BIT_LED );
```

The following statement is used to read the content of **gpio_ext_porta** register. The bit mask is used to check the status of the key.

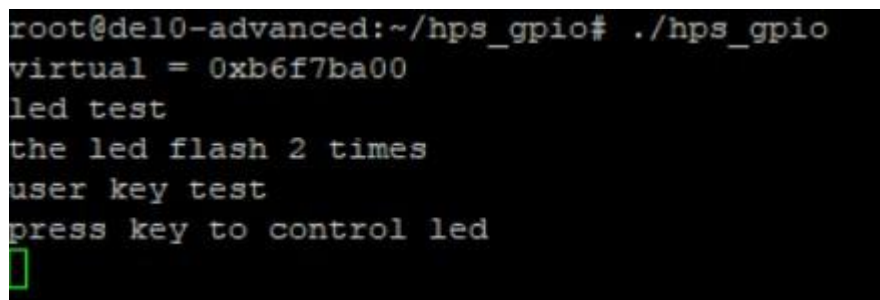
```
alt_read_word( virtual_base + ALT_GPIO_EXT_PORTA_OFST );
```

■ Demonstration Source Code

- Build tool: SoC EDS V18.0
- Project directory: \Demonstration\SoC\hps_gpio
- Binary file: hps_gpio
- Build command: make ('make clean' to remove all temporal files)
- Execute command: ./hps_gpio

■ Demonstration Setup

1. Connect a USB cable to the USB-to-UART connector (J27) on the HAN Pilot Platform and the host PC.
2. Copy the executable file "**hps_gpio**" into the microSD card under the **"/home/root"** folder in Linux.
3. Insert the booting micro SD card into the HAN Pilot Platform.
4. Power on the HAN Pilot Platform.
5. Launch Putty and establish connection to the UART port of Putty. Type **"root"** to login into Linux.
6. Type **"./hps_gpio"** in the UART terminal of Putty to start the program, as shown in **Figure 3-6**.



```
root@del0-advanced:~/hps_gpio# ./hps_gpio
virtual = 0xb6f7ba00
led test
the led flash 2 times
user key test
press key to control led
█
```

Figure 3-6 Start the Program

7. HPS_LED will flash twice and users can control the user LED with push-button.
8. Press HPS_KEY to light on and off the HPS_LED.
9. Press "CTRL + C" to terminate the application.

3.2 Setup USB Wi-Fi Dongle

This section describes how to setup the Wi-Fi USB dongle under Linux, so Linux user can wirelessly connect to the Wi-Fi AP (Access Point) through the Wi-Fi USB Dongle and finally connect to the internet. The Wi-Fi AP is assumed to have the DHCP server capability and is connected to the internet. You should also make sure you know the SSID and Password of the Wi-Fi AP.

■ System Diagram

Figure 3-7 shows the block diagram of this demonstration. The Wi-Fi AP assumes you have the DHCP server capability and is connected to the LAN (Local Area Network) or the internet. The USB Wi-Fi Dongle connects to the Wi-Fi AP and gets an address IP from the Wi-Fi AP. Through the Wi-Fi AP, the USB-Dongle will be able to communicate with the devices connected to the LAN or the internet.

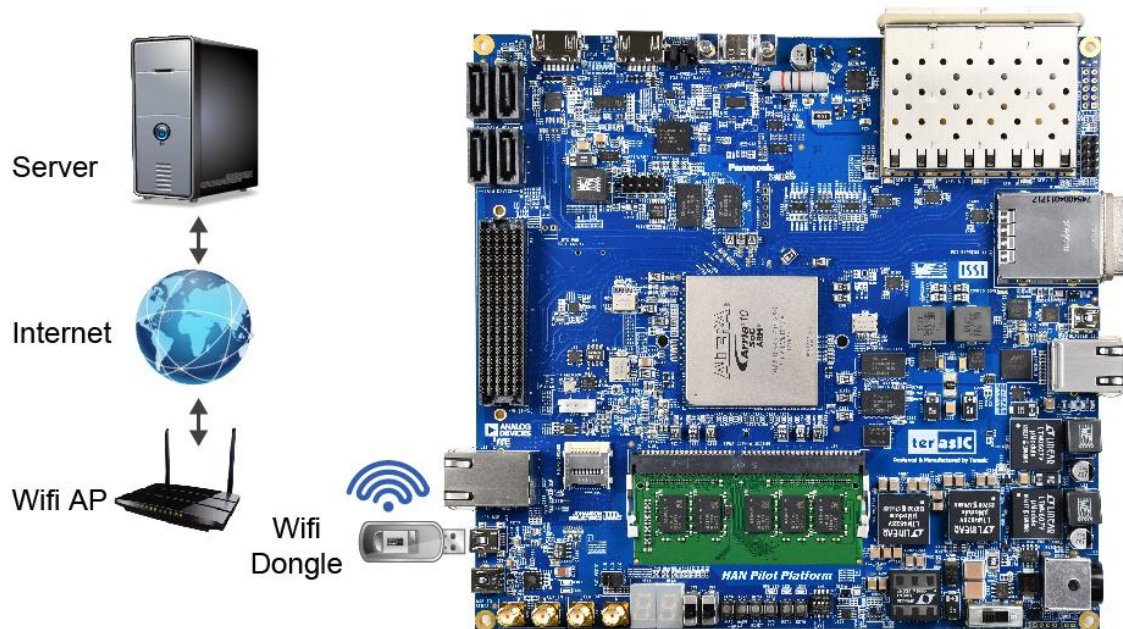


Figure 3-7 System diagram of USB Wi-Fi dongle

■ Wi-Fi Setup Procedure

1. Connect a USB cable to the USB-to-UART connector (J27) on the HAN Pilot Platform and the host PC.
2. Connect the USB Wi-Fi Dongle into the USB port on the HAN Pilot Platform with USB Transfer Cable.
3. Power on the HAN Pilot Platform.
4. Launch Putty to establish the connection between the UART port of the HAN Pilot Platform and the host PC. Type "root" and then press "Enter." By pressing "Enter," you can login to Linux without password.
5. Type **"ifconfig wlan0 up"** in the UART terminal of Putty to start wlan0 network interface.
6. Type **"iwlist wlan0 scan | grep ESSID"** in the UART terminal to search nearby Wi-Fi AP. Make sure your Wi-Fi AP is found, as shown in **Figure 3-8**.

```

root@del0-advanced:~# ifconfig wlan0 up
root@del0-advanced:~# iwlist wlan0 scan | grep ESSID
ESSID:"[REDACTED]"
ESSID:"[REDACTED]"
ESSID:"[REDACTED]"
ESSID:"[REDACTED]"
ESSID:"[REDACTED]"
ESSID:"Terasic"

```

Figure 3-8 Wi-Fi AP information

7. Type "**vim /etc/wpa_supplicant/wpa_supplicant.conf**" in the UART terminal to edit Wi-Fi configuration file, as shown in **Figure 3-9**.

```

ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="Your_SSID"
    psk="Your_WPA-Key_ASCII"
}
~

```

Figure 3-9 Edit Wi-Fi configuration File

8. In the configuration file, replace "Your_SSID" and "Your_WPA-Key_ASCII" with the SSID and password for your Wi-Fi AP, in respectively, as shown in **Figure 3-10**.

```

ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="Terasic"
    psk="1234567890"
}
~

```

Figure 3-10 Replace ssid and psk

9. Type "**ifup wlan0**" in the UART terminal to connect to the Wi-Fi AP, as shown in **Figure 3-11**.
10. Type "**ifconfig wlan0**" in the UART terminal to confirm an IP Address is assigned to wlan0 interface, as shown in **Figure 3-12**.
11. Make sure Wi-Fi AP is connected to the internet. Type "**ping -c 4 www.terasic.com**" in the UART terminal to check internet connection status. If 0% packet loss is reported, it means the connection is good, as shown in **Figure 3-13**.

```

root@del0-advanced:~# ifup wlan0
Internet Systems Consortium DHCP Client 4.3.3
Copyright 2004-2015 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlan0/fc:3d:93:b9:18:6d
Sending on   LPF/wlan0/fc:3d:93:b9:18:6d
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 3 (xid=0xc8f8d06a)
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 8 (xid=0xc8f8d06a)
DHCPCREQUEST of 192.168.43.249 on wlan0 to 255.255.255.255 port 67 (xid=0x6ad0f8c8)
DHCPOFFER of 192.168.43.249 from 192.168.43.1
DHCPACK of 192.168.43.249 from 192.168.43.1
bound to 192.168.43.249 -- renewal in 1457 seconds.
root@del0-advanced:~# █

```

Figure 3-11 Type "ifup wlan0"

```

root@del0-advanced:~# ifconfig wlan0
wlan0      Link encap:Ethernet  HWaddr fc:3d:93:b9:18:6d
           inet addr:192.168.43.249  Bcast:192.168.43.255  Mask:255.255.255.0
           inet6 addr: fe80::fe3d:93ff:feb9:186d/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:9 errors:0 dropped:0 overruns:0 frame:0
           TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:1535 (1.5 KB)  TX bytes:2132 (2.1 KB)

root@del0-advanced:~# █

```

Figure 3-12 Type "ifconfig wlan0"

```

root@del0-advanced:~# ping -c 4 www.terasic.com
PING www.terasic.com (74.207.250.186) 56(84) bytes of data.
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=1 ttl=51 time=218 ms
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=3 ttl=51 time=242 ms
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=4 ttl=51 time=221 ms

--- www.terasic.com ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 3010ms
rtt min/avg/max/mdev = 218.687/227.683/242.571/10.617 ms
root@del0-advanced:~# █

```

Figure 3-13 Type "ping -c 4 www.terasic.com"

3.3 HPS GPIO Header

This demonstration shows how to use the system call with built-in GPIO driver to implement HPS GPIO Header's loopback. The built-in GPIO driver is included the HAN Pilot Platform LXDE VNC Desktop BSP.

■ Function Block Diagram

Figure 3-14 shows the function block diagram of the HPS GPIO Header loopback demonstration. The built-in GPIO driver offers interfaces, to which the application can use system call such as open, read, write to access. We can export the gpio port that we want to control, and when we export the

gpio port, the linux system will create attribute files of the gpio port in the location “/sys/class/gpio/gpioN/” (N is the gpio port’ number). There are two attribute files we need to know: value and direction. The value file is used to read and write value to the gpio port (the value can only be “0” or “1”); the direction file is used to set the gpio port’s data direction.

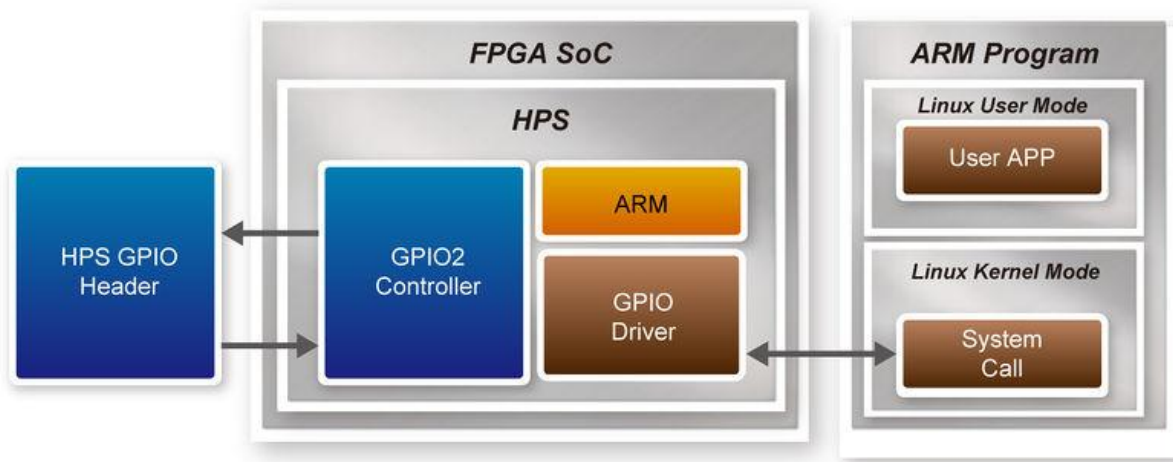


Figure 3-14 Function block diagram of HPS GPIO Header demonstration

■ Function Implement

In the c code project, we need to implement five functions, described as following:

int gpio_export(unsigned int gpio);

The gpio_export function is used to export the gpio port with the specified port number as parameter.

int gpio_unexport(unsigned int gpio);

The gpio_unexport function is used to disable the exported gpio port with the specified port number as parameter.

int gpio_set_dir(unsigned int gpio, unsigned int out_flag);

The gpio_set_dir function is used to set the gpio port’s data direction, the parameter “gpio” is the port number you want to configure and the parameter “out_flag” is value to set. Number “1” for data out, and “0” for data in. when you use this api, it will write “in” or “out” to the gpio port’s direction file. The default value of direction file is “in”.

int gpio_set_value(unsigned int gpio, unsigned int value);

The gpio_set_value function is used to write data to the gpio port. The parameter “gpio” is the port number you want to configure and then parameter “value” is the data you want to write. The value can only be “0” or “1”. When you use the api, it will write data to the gpio port’s value file.

int gpio_get_value(unsigned int gpio, unsigned int *value);

The gpio_get_value function is used to read the gpio port’s data, and the parameter “value” is used to store the value that you read. The parameter “gpio” is the gpio port that you want to read.

All the functions are implemented in the c code file, you can get more details from the c code file.

■ Loopback Implement

There are four gpio ports used to loopback. They are IO8, IO9, IO10, IO11. The Loopback includes two test patterns, the differences between them are data direction and test data value. In test one, we set IO 8 up to 11 as “out”, “in”, “out”, “in” respectively, and the test data is a 32-bit value “0x1234f0f0”.

Described below are the loopback’s implementation procedure:

- Export gpios
- Set gpios’s data direction
- Data write and read back
- Verify the received data

Figure 3-15 shows the procedure in c code, you can find it’s very clear.

```
// export gpio2's io8 io9 io10 io11
gpio_export(io8);
gpio_export(io9);
gpio_export(io10);
gpio_export(io11);

printf("\n=====Loopback Test:Start Test1=====\\n");
printf("Test 1 : io8->io9,io10->io11, write data: 0x%x\\n", test_data1);

// set direction(0:in 1:out) : 11:in, 10:out, 9:in, 8:out
gpio_set_dir(io8, 1);
gpio_set_dir(io9, 0);
gpio_set_dir(io10, 1);
gpio_set_dir(io11, 0);

// data write and read back
for (i=0; i<32; i++){
    tmp_data = (unsigned int)((test_data1>>(31-i))&0x1);

    gpio_set_value(io8,tmp_data);
    gpio_set_value(io10,tmp_data);

    gpio_get_value(io9,&tmp_data);
    rcv_data1 = (rcv_data1<<1)|tmp_data;

    gpio_get_value(io11,&tmp_data);
    rcv_data2 = (rcv_data2<<1)|tmp_data;
}

printf("Recv Data : IO9 = 0x%x\\n", rcv_data1);
printf("Recv Data : IO11 = 0x%x\\n", rcv_data2);

// verify1
if (rcv_data1==test_data1){
    printf("Test1 IO8->IO9 successfully\\n");
}else
    printf("Test1 IO8->IO9 failed\\n");

if (rcv_data2==test_data1){
    printf("Test1 IO10->IO11 successfully\\n");
}else
    printf("Test1 IO10->IO11 failed\\n");
```

Figure 3-15 loopback implemented in c code

■ Demonstration Source Code

- Build tool: SoC EDS V18.0
- Project directory: \Demonstration\SoC\hps_gpio_loopback
- Binary file: hps_gpio_loopback
- Build command: make ('make clean' to remove all temporal files)

- Execute command: `./hps_gpio_loopback`

■ Demonstration Setup

1. Use jumper cap connect IO8 to IO9 and IO10 to IO11 in hps gpio header(J36) on the HAN Pilot Platform. **Figure 3-16** shows the pin location below.

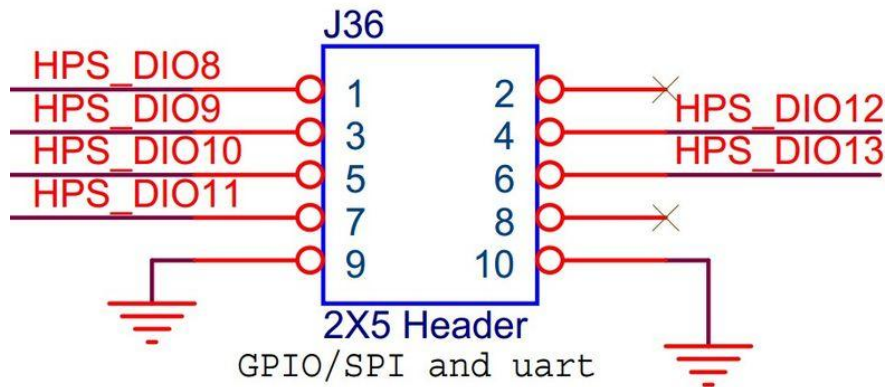


Figure 3-16 GPIO Header Pin location

2. Connect a USB cable to the USB-to-UART connector (J27) on the HAN Pilot Platform and the host PC.
3. Copy the executable file "**hps_gpo_loopback**" into the microSD card under the **"/home/root"** folder in Linux. (HAN Pilot Platform LXDE has pre-installed this code, so users can skip this copy action.)
4. Insert the LXDE booting micro SD card into the HAN Pilot Platform.
5. Power on the HAN Pilot Platform.
6. Launch Putty to establish the connection between the UART port of HAN Pilot Platform and the host PC.
7. In the Putty UART terminal, type "root" to login LXDE Linux with pressing Enter as password.
8. Type "**cd hps_gpio_loopback**" to into the folder and type "**./hps_gpio_loopback**" in the UART terminal to start the program.
9. You will see the loopback test successfully in the Putty UART terminal as shown in **Figure 3-17**.

```

root@del0-advanced:~/hps_gpio_loopback# ./hps_gpio_loopback

=====Loopback Test:Start Test1=====
Test 1 : io8->io9,io10->io11, write data: 0x1234f0f0
Recv Data : IO9 = 0x1234f0f0
Recv Data : IO11 = 0x1234f0f0
Test1 IO8->IO9 successfully
Test1 IO10->IO11 successfully

=====Loopback Test:Start Test2=====
Test 2 : io8<-io9,io10<-io11, write data: 0x43210f0f
Recv Data : IO8 = 0x43210f0f
Recv Data : IO10 = 0x43210f0f
Test2 IO8<-IO9 successfully
Test2 IO10<-IO11 successfully
root@del0-advanced:~/hps_gpio_loopback# █

```

Figure 3-17 Loopback test successfully

3.4 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 3-18](#). 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.

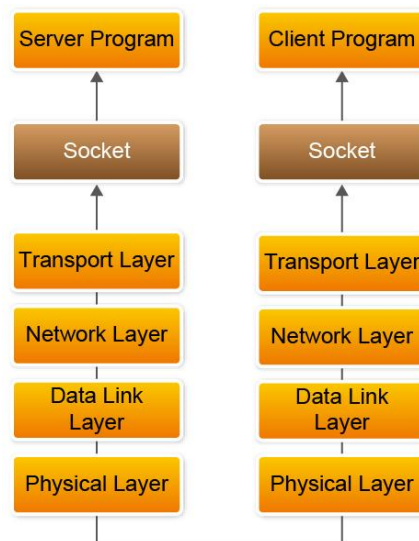


Figure 3-18 Communicate on a network via a socket

■ 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 3-19** shows the communication diagram between the client and server.

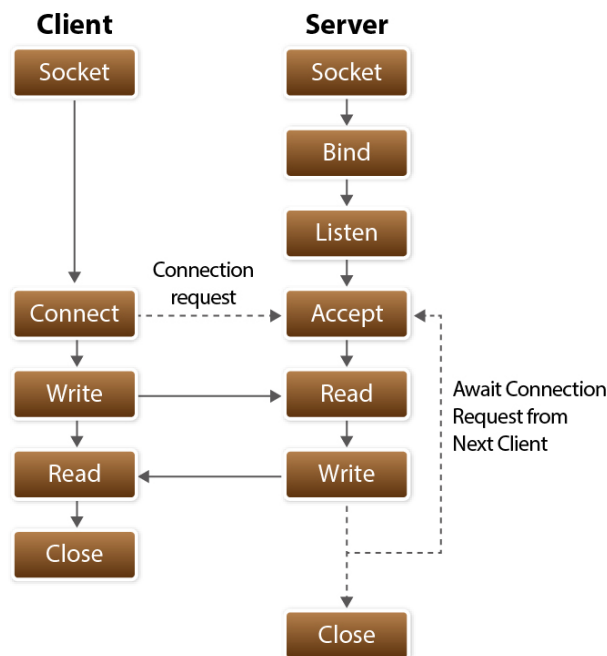


Figure 3-19 Client and Server communication

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 3-20** 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 3-20 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 3-21** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK_STREAM socket, **connect** API is used to connect the remove socket sever based on the given hostname (or IPv4 Address) and port number. Data receiving and sending is implemented by **read** and **write** API, and **close** is used to **close** the socket.

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr, "ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr.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 3-21 Socket Client Code

■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-22**. The Demonstration folder contains three platform subfolders: **arm**, **linux** and **windows**. The project under the **arm** folder is designed for SoC FPGA board. The project under **linux** folder is designed for Linux running on Linux PC. The project under **windows** folder is designed for SoC EDS Shell running on Windows PC. Each platform subfolder contains **socket_client** and **socket_server** project folders.

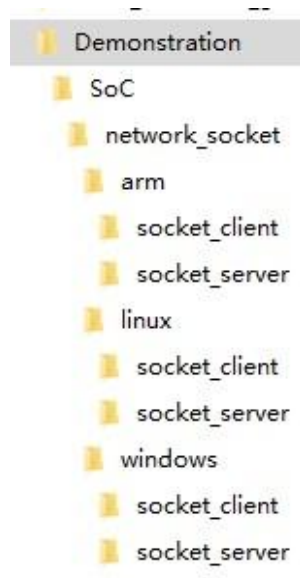


Figure 3-22 Source Code Folder Tree

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 HAN Pilot Platform to Network via Ethernet port (J25).
2. Connect a USB cable to the USB-to-UART connector (J27) on the HAN Pilot Platform and the host Windows PC.
3. Copy the executable file “socket_server” into the microSD card under the **"/home/root"** folder in Linux. (HAN Pilot Platform LXDE has pre-installed this code, so users can skip this copy action.)
4. Insert the LXDE booting micro SD card into the HAN Pilot Platform.
5. Power on the HAN Pilot Platform.
6. In Windows, launch the Putty to connect HAN Pilot Platform via the UART-to-USB port.
7. In the Putty, type "root" to login LXDE Linux.
8. Type **"./socket_server 2020"** to launch the server program with port number 2020 as shown in **Figure 3-23**. The port number can be any value between 2000 and 63500.

```
root@del10-advanced:~/socket_server# ./socket_server 2020
```

Figure 3-23 Start Socket Server

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

1. Make sure the SoC EDS is installed on your Windows and the Windows is connected to a network.
2. Copy the client program (windows/socket_client/socket_client.exe) in the example kit to your Windows.
3. Launch the SoC EDS Command Shell.
4. In the command shell, change the current directory to the directory where socket_client.exe 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 3-24**.

```
btdai@daibitao /cygdrive/d/workspace/del0_advanced_revC/Demonstration/SoC/network_socket/windows/socket_client
$ ./socket_client.exe 192.168.2.27 2020
```

Figure 3-24 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 3-25**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-26**.

```
btdai@daibitao /cygdrive/d/workspace/del0_advanced_revC/Demonstration/SoC/network_socket/windows/socket_client
$ ./socket_client.exe 192.168.2.27 2020
Please enter the message: hello
I got your message
btdai@daibitao /cygdrive/d/workspace/del0_advanced_revC/Demonstration/SoC/network_socket/windows/socket_client
$
```

Figure 3-25 Send Message in Client Program

```
root@del0-advanced:~/socket_server# ./socket_server 2020
Here is the message: hello

root@del0-advanced:~/socket_server#
```

Figure 3-26 Server dumps received message

Chapter 4

Examples for Using both HPS SoC and FPGA

This Chapter demonstrates how to use the HPS/ARM to communicate with FPGA. We will introduce the GHRD project for HAN Pilot Platform development board. And we develop one ARM C Project which demonstrates how HPS/ARM program controls the two 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 operation skill
- Basic RTL coding skill
- Basic Qsys operation skill
- Knowledge about Memory-Mapped Interface

■ C Program Design

- Basic SoC EDS (Embedded Design Suite) operation skill
- Basic C coding and compiling skill
- Skill to Create a Linux Boot SD-Card for HAN Pilot Platform with a given image file
- Skill to boot Linux from SD-Card on HAN Pilot Platform Skill to copy files into Linux file system on HAN Pilot Platform Basic Linux command operation skill

4.2 System Requirements

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

■ Terasic HAN Pilot Platform, includes

- Mini USB Cable for UART terminal
- Micros SD-Card, at 4GB minimum
- Micros SD-Card Card Reader

■ A x86 PC

- Windows 10 64bit operation system Installed
- One USB Port
- Quartus Prime 18.0 or Later Installed
- SoC EDS 18.0 or Later Installed
- Win32 Disk Imager Installed

4.3 AXI bridges in Intel SoC FPGA

In Intel 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, Intel 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 Qsys 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 of the AXI bridges in the context of the FPGA fabric and the L3 interconnect to the HPS. Each master (M) and slave (S) interface is shown with its data width(s). The clock domain for each interconnect is noted in parentheses.

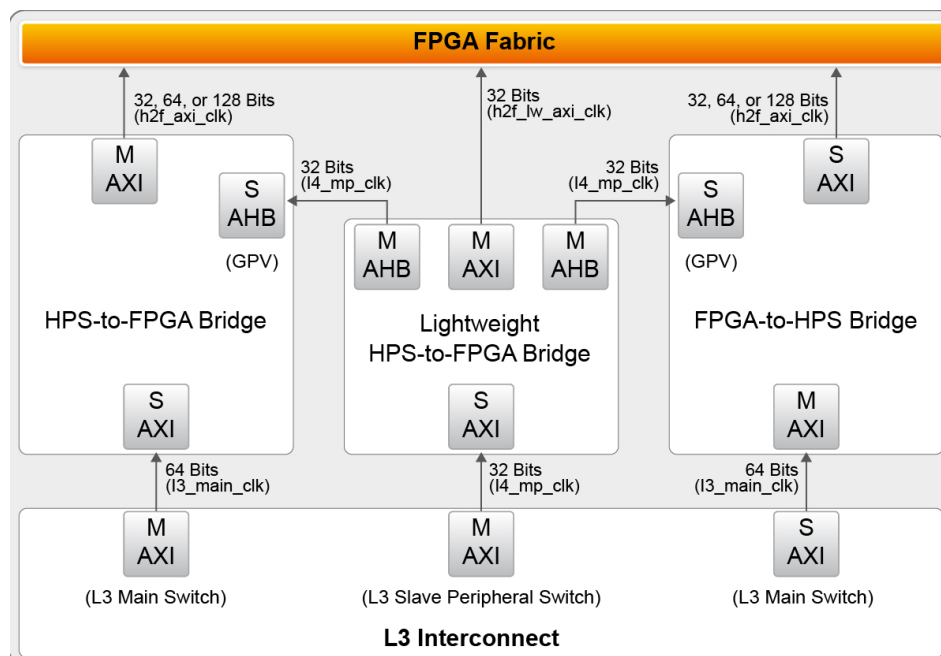


Figure 4-1 AXI Bridge Block Diagram

The HPS-to-FPGA bridge is mastered by the level 3 (L3) main switch and the lightweight

HPS-to-FPGA bridge is mastered by the L3 slave peripheral switch.

The FPGA-to-HPS bridge masters the L3 main switch, allowing any master implemented in the FPGA fabric to access most slaves in the HPS. For example, the FPGA-to-HPS bridge can access the accelerator coherency.

All three bridges contain global programmer view GPV register. The GPV register control the behavior of the bridge. It is able to access to the GPV registers of all three bridges through the lightweight HPS-to-FPGA bridge.

This Demo introduces to users how to use the HPS/ARM to communicate with FPGA. This project includes GHRD project for the HAN Pilot Platform one ARM C Project which demonstrates how HPS/ARM program controls the LEDs connected to FPGA.

4.4 GHRD Project

The term GHRD is short for Golden Hardware Reference Design. The GRD project provide by Terasic for the HAN Pilot Platform is located in the CD folder: CD-ROM\ Demonstration\ SOC_FPGA\ a10s_ghrd.

The project consists of the following components:

- ARM Cortex™-A9 MP Core HPS
- Two user push-button inputs
- Two user DIP switch inputs
- Two user I/O for LED outputs
- 256 KB of on-chip memory
- JTAG to Avalon master bridges
- Interrupt capturer for use with System Console
- System ID

The memory map of system peripherals in the FPGA portion of the SoC as viewed by the MPU starts at the lightweight HPS-to-FPGA base address 0xFF20_0000. The MPU can access these peripherals through the Address offset setting in the Qsys. User can open the GHRD project with Quartus Software. Then open the a10s_ghrd.qsys file with the Qsys tool. **Figure 4-2** lists the address map of the peripherals which are connected to the lightweight HPS-to-FPGA.

System	Path	Address Map
del0_advanced_ghrd	Path: arria10_hps.h2f_lw_axi_master	
.....hps_m.master	pb_lwh2f.m0	
ILC_avalon_slave		0x0000 - 0x00ff
arria10_hps.f2sdram0...		
arria10_hps.f2h_axi_s...		0x0000 0000 - 0xffff ffff
button_pio.sl		0x0110 - 0x011f
dipsw_pio.sl		0x0100 - 0x010f
led_pio.sl		0x0120 - 0x012f
onchip_memory2.sl		
pb_lwh2f.s0		
sysid_qsys.control_slave		0x0130 - 0x0137
ILC_avalon_slave via ...		
sysid_qsys.control_sl...		
led_pio.sl via pb_lwh2f		
dipsw_pio.sl via pb_l...		
button_pio.sl via pb...		

Figure 4-2 FPGA peripherals address map

All the Avalon Conduit signals of these peripherals are connected to the I/O pins of the SoC FPGA on HAN Pilot Platform as shown in the **Figure 4-3**.

<code>// UART</code>			
<code>.hps_io_hps_io_phery_uart1_RX</code>	<code>(HPS_RXD),</code>	<code>//</code>	<code>.hps_io_phery_1</code>
<code>.hps_io_hps_io_phery_uart1_TX</code>	<code>(HPS_TXD),</code>	<code>//</code>	<code>.hps_io_phery_1</code>
<code>// I2C</code>			
<code>.hps_io_hps_io_phery_i2c0_SDA</code>	<code>(HPS_I2C0_SDAT),</code>	<code>//</code>	<code>.hps_io_phery_1</code>
<code>.hps_io_hps_io_phery_i2c0_SCL</code>	<code>(HPS_I2C0_SCLK),</code>	<code>//</code>	<code>.hps_io_phery_1</code>
<code>// GPIO</code>			
<code>.hps_io_hps_io_gpio_gpio1_io1</code>	<code>(HPS_LED),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.hps_io_hps_io_gpio_gpio1_io4</code>	<code>(HPS_KEY),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.hps_io_hps_io_gpio_gpio2_io8</code>	<code>(HPS_GPIO[0]),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.hps_io_hps_io_gpio_gpio2_io9</code>	<code>(HPS_GPIO[1]),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.hps_io_hps_io_gpio_gpio2_io10</code>	<code>(HPS_GPIO[2]),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.hps_io_hps_io_gpio_gpio2_io11</code>	<code>(HPS_GPIO[3]),</code>	<code>//</code>	<code>.hps_io_gpio_gp</code>
<code>.led_pio_external_connection_export</code>	<code>(LED),</code>	<code>//</code>	<code>led_pio_external_connection.export</code>
<code>.button_pio_external_connection_export</code>	<code>(fpga_debounced_buttons),</code>	<code>//</code>	<code>button_pio_external_connection.export</code>
<code>.dipsw_pio_external_connection_export</code>	<code>(SW),</code>	<code>//</code>	<code>dipsw_pio_external_connection.export</code>

Figure 4-3 Connection in the top design

4.5 Compile and Programming

In the Platform Design tool, click the menu item “Generate Generate...” to generate source code for the system and then close the Qsys tool. Now, users can start the compile process by clicking the menu item “Processing Start Compilation”.

When the compilation process is completed successfully, **a10s.sof** is generated in the `a10s_ghrd/output_files` folder. Users can use this file to configure FPGA by Quartus Programming through the HAN Pilot Platform on-board USB-Blaster II.

4.6 Develop the C Code

This section introduces how to design an ARM C program to control the **led_pio** PIO controller. SoC EDS 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.

■ LED_PIO Address

The led_pio component information is required for ARM C program as the program will attempt to control the component. This section describes how to get led_pio’s address.

You can get led_pio’s address from qsys’s Address Map dialog box. **Figure 4-4** shows led_pio’s address in Address Map. You can define a macro for the address when you use it.

System Contents			Address Map	Interconnect Requirements
System: de10_advanced_ghrd			Path: arria10_hps.h2f_lw_axi_master	
hps_m.master		pb_lwh2f.m0	
ILC_avalon_slave			0x0000 - 0x00ff	
arria10_hps.f2sdram0...				
arria10_hps.f2h_axi_s...			0x0000 0000 - 0xffff ffff	
button_pio.s1			0x0110 - 0x011f	
dipsw_pio.s1			0x0100 - 0x010f	
led_pio.s1			0x0120 - 0x012f	
onchip_memory2.s1				
pb_lwh2f.s0				
sysid_qsys.control_slave			0x0130 - 0x0137	
ILC_avalon_slave via ...				
sysid_qsys.control_sl...				
led_pio.s1 via pb_lwh2f				
dipsw_pio.s1 via pb_l...				
button_pio.s1 via pb...				

Figure 4-4 PIO led address in Qsys's Address Map

■ Map LED_PIO Address

This section will describe how to map the led_pio 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 (HW_REGS_BASE = 0xfc000000) 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 of led_pio can be calculated by adding the below two offset addresses to virtual_base.

- Offset address of Lightweight HPS-to-FPGA AXI bus relative to HPS base address
- Offset address of Pio_led relative to Lightweight HPS-to-FPGA AXI bus

The first offset address is 0xff200000 which is defined as a constant ALT_FPGA_BRIDGE_LWH2F_OFST in the header hps.h. The hps.h is a header of SoC EDS. It is located in the Quartus installation folder: D:\IntelFPGA\18.0\embedded\ip\altera\hps\altera_hps\hwlib\include\soc_a10\socal.

The second offset address is 0x120 which is led_pio's address defined as LED_PIO_BASE in the C code file.

The virtual address of led_pio is represented by a void pointer variable **h2p_lw_led_addr**. Application program can directly use the pointer variable to access the registers in the controller of **LED_PIO**.

```

#define HW_REGS_BASE ( ALT_FPGA_BRIDGE_LWH2F_OFST )
#define HW_REGS_SPAN ( 0x200000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

#define LED_PIO_BASE (0x120)
#define LED_PIO_DATA_WIDTH 2

int main() {
    void *virtual_base;
    int fd;
    int loop_count;
    int led_direction;
    int led_mask;
    void *h2p_lw_led_addr;

    // 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("HW_REGS_BASE=0x%x\n",HW_REGS_BASE);
    virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd, HW_REGS_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 ) ( ALT_FPGA_BRIDGE_LWH2F_OFST + LED_PIO_BASE ) & ( unsigned long )

```

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. Because the led on HAN Pilot Platform is low active, writing a value 0x00000003 to the offset 0 register will turn off the two LEDs. Writing a value 0x00000000 to the offset 0 register will turn on the two LEDs. In C program, writing a value 0x00000000 to the offset 0 register of 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	Fields				
				(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.				
5	outclear		W	Specifies which output bit to clear.				

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.

```
// toggle the LEDs a bit

loop_count = 0;
led_mask = 0x01;
led_direction = 0; // 0: left to right direction
while( loop_count < 60 ) {

    // control led
    *(uint32_t *)h2p_lw_led_addr = ~led_mask;

    // wait 100ms
    usleep( 100*1000 );

    // update led mask
    if (led_direction == 0){
        led_mask <<= 1;
        if (led_mask == (0x01 << (LED_PIO_DATA_WIDTH-1)))
            led_direction = 1;
    }else{
        led_mask >>= 1;
        if (led_mask == 0x01){
            led_direction = 0;
            loop_count++;
        }
    }
} // while
```

Figure 4-7 C Program for LED Shift Operation

■ Makefile and compile

Figure 4-8 shows the content of Makefile for this C project. The program includes the head files provided by SoC EDS. In the Makefile, ARM-linux cross-compile also be specified.

```
#
TARGET = hps_fpga_led

#
ALT_DEVICE_FAMILY ?= soc_a10
SOCEDS_ROOT ?= $(SOCEDS_DEST_ROOT)
HWLIBS_ROOT = $(SOCEDS_ROOT)/ip/altera/hps/altera_hps/hwlib
CROSS_COMPILE = arm-linux-gnueabi-
CFLAGS = -g -Wall -D$(ALT_DEVICE_FAMILY) -I$(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY) -I$(HWLIBS_ROOT)/include/
LDFLAGS = -g -Wall
CC = $(CROSS_COMPILE)gcc
ARCH= arm

build: $(TARGET)
$(TARGET): main.o
    $(CC) $(LDFLAGS) $^ -o $@
%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(TARGET) *.a *.o *~
```

Figure 4-8 Makefile content

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”.

```

btdai@daibitao /cygdrive/d/workspace/de10_advanced_revC/will/hps_fpga_led
$ ls
main.c  Makefile

btdai@daibitao /cygdrive/d/workspace/de10_advanced_revC/will/hps_fpga_led
$ make
arm-linux-gnueabi-gcc -g -Wall -Dsoc_al0 -ID:/softwares/IntelFPGA/18.0/embedded/ip/altera/hps/altera_hps/hwlib/i
de/soc_al0 -ID:/softwares/IntelFPGA/18.0/embedded/ip/altera/hps/altera_hps/hwlib/include/ -c main.c -o main.o
main.c: In function 'main':
main.c:39:3: warning: implicit declaration of function 'strerror' [-Wimplicit-function-declaration]
printf( "errno : %s\n", strerror(errno) );

main.c:39:3: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'int' [-Wformat=]
arm-linux-gnueabi-gcc -g -Wall main.o -o hps_fpga_led

btdai@daibitao /cygdrive/d/workspace/de10_advanced_revC/will/hps_fpga_led
$ ls
hps_fpga_led  main.c  main.o  Makefile

btdai@daibitao /cygdrive/d/workspace/de10_advanced_revC/will/hps_fpga_led
$

```

Figure 4-9 ARM C Project Compilation

■ Execute the Demo

To execute the demo, please boot the Linux from the SD-card in HAN Pilot Platform. 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. Then, type “./ hps_fpga_led” to launch the ARM program. The LED[1:0] on HAN Pilot Platform will be expected to perform 60 times of LED light shift operation, and then the program is terminated.

For details about booting the Linux from SD-card, please refer to the document: ***Getting_Started_Guide.pdf***

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 Windows 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](#).

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

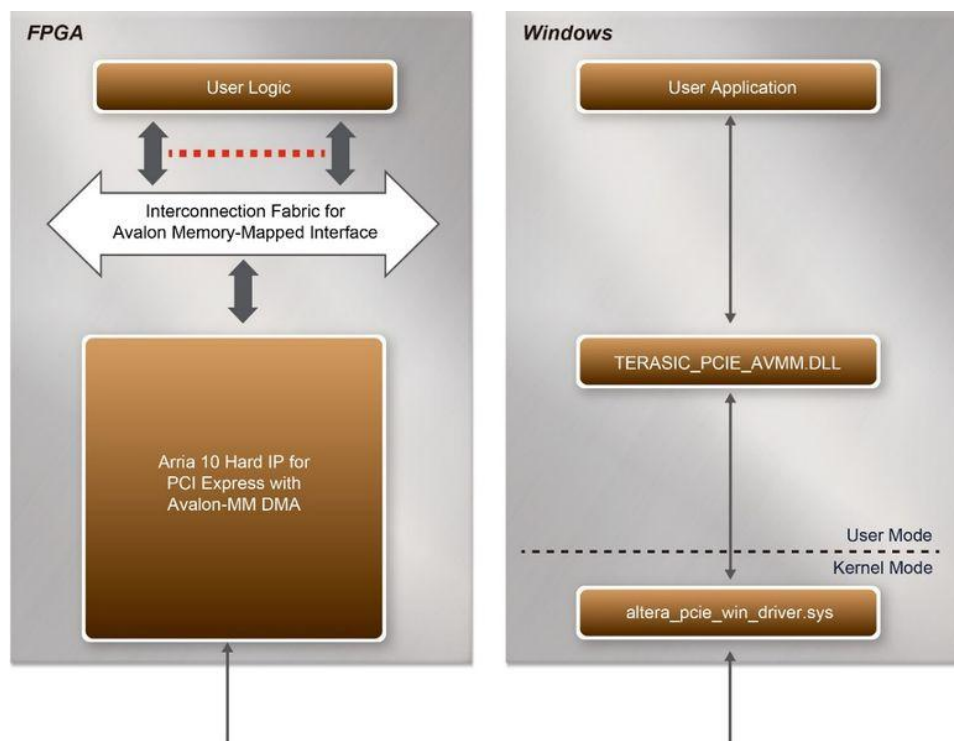


Figure 5-1 Infrastructure of PCI Express System

5.2 PC PCI Express Software SDK

The FPGA System CD contains a PC 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 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.

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_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.

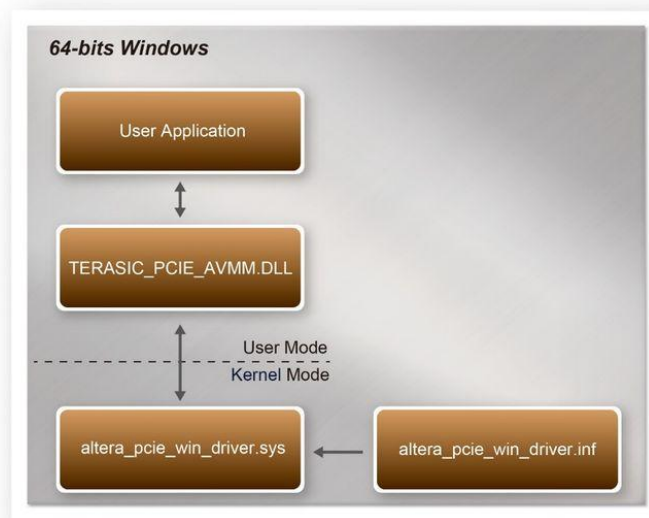


Figure 5-2 PCI Express Software Stack

■ Install PCI Express Driver on Windows

The PCIe driver is located 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. Make sure the HAN Pilot Platform and the PC are both powered off.
2. Set MSEL[2:0] to 010.
3. Plug the PCIe adapter card into the PCIe slot on the PC motherboard. Use the PCIe cable to connect to the HAN Pilot Platform PCIe connector and the PCIe adapter card (See **Figure 5-3**)

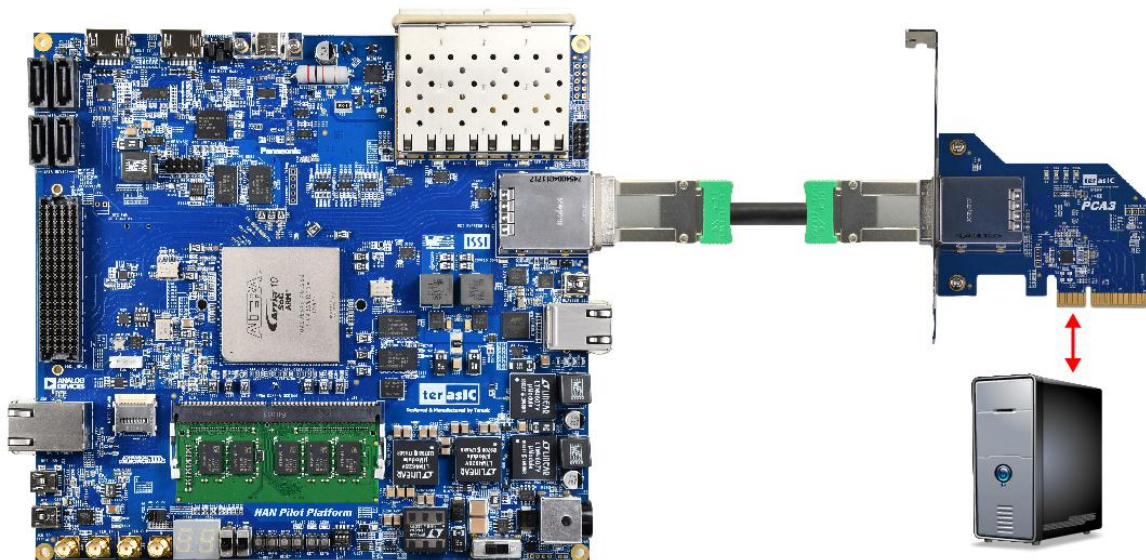


Figure 5-3 FPGA board connect to PC

4. Power on your HAN Pilot Platform and the host PC.
5. Make sure Altera Programmer and USB-Blaster II driver are installed
6. Execute test.bat in "CDROM\Demonstrations\PCIe_Fundamental\demo_batch" to configure the FPGA
7. Restart windows operation system
8. 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 5-4**. Move the mouse cursor to the PCI Device item and right click it to select the Update Driver Software... item.

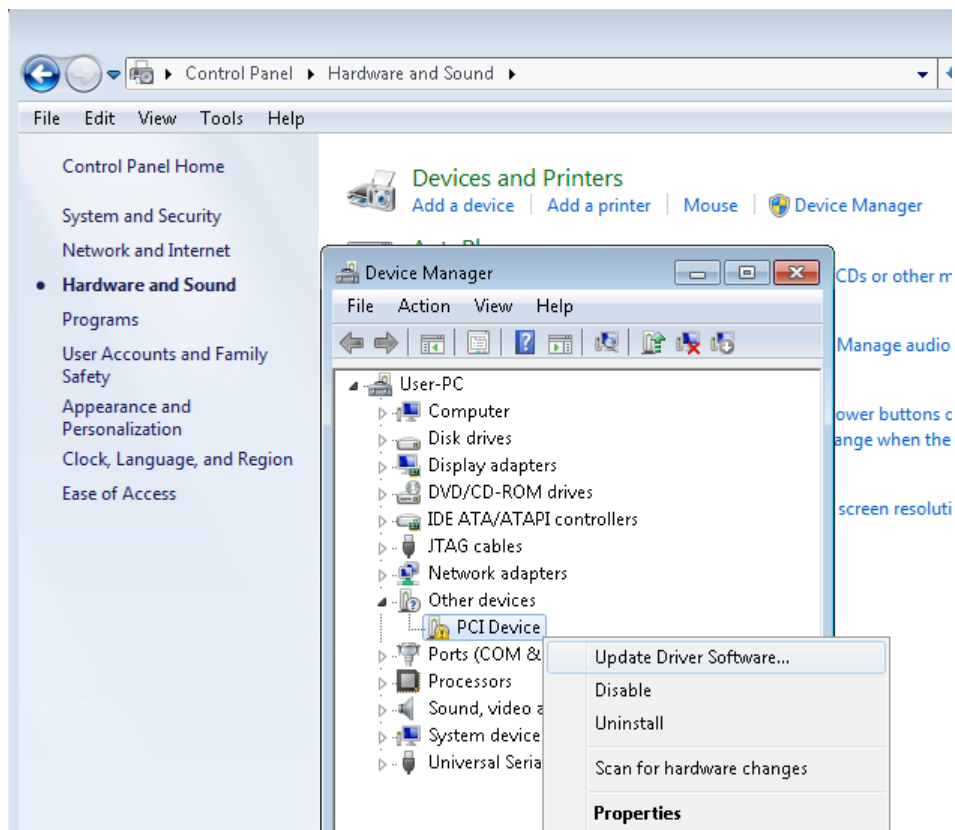


Figure 5-4 Screenshot of launching Update Driver Software... dialog

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

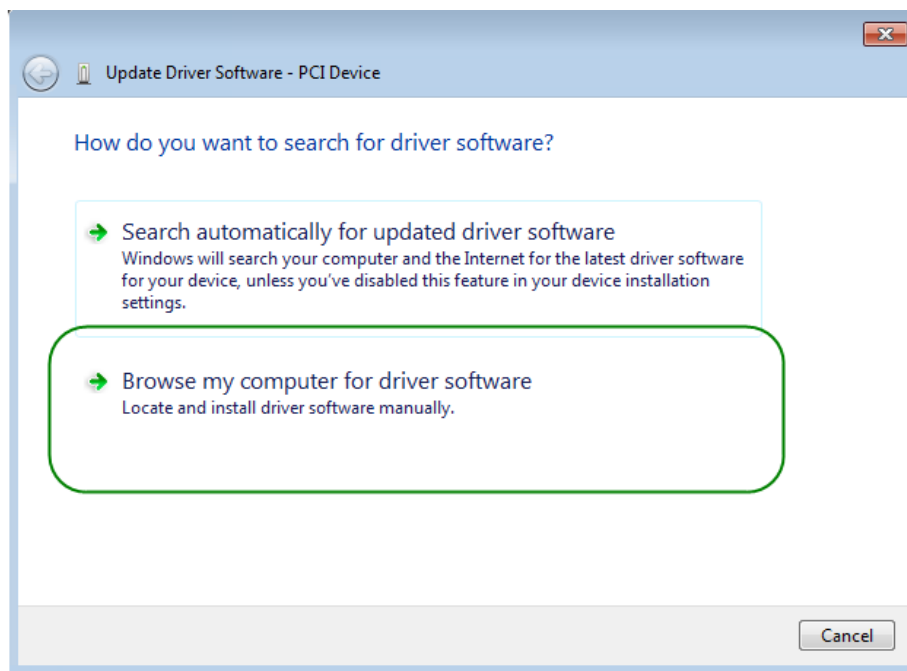


Figure 5-5 Dialog of Browse my computer for driver software

10. 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 5-6**. Click the Next button

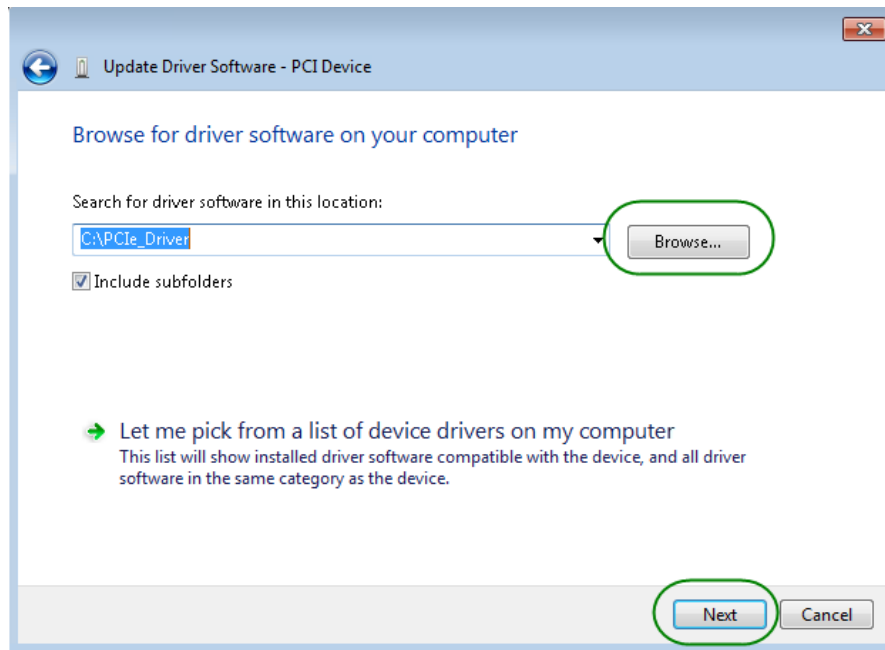


Figure 5-6 Browse for driver software on your computer

11. When the Windows Security dialog appears, as shown **Figure 5-7**, click the Install button.

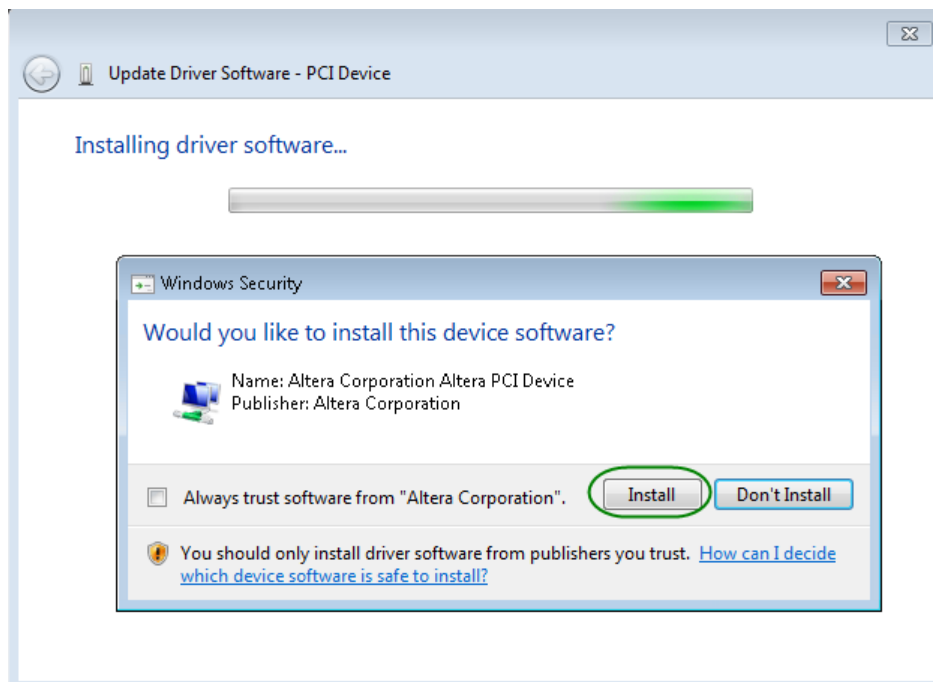


Figure 5-7 Click Install in the dialog of Windows Security

12. When the driver is installed successfully, the successfully dialog will appear, as shown in **Figure 5-8**. Click the Close button.

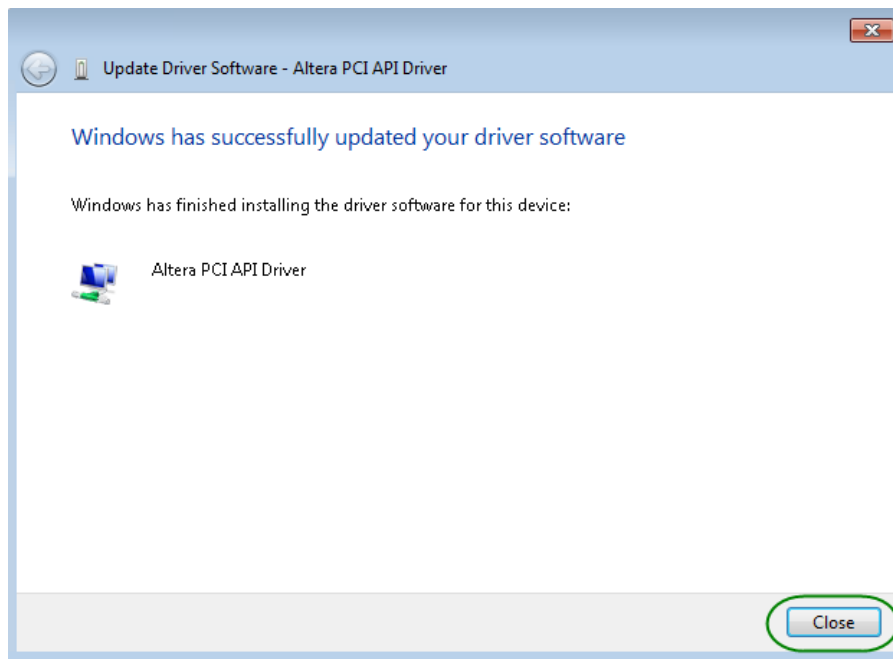


Figure 5-8 Click Close when the installation of Altera PCI API Driver is complete

13. Once the driver is successfully installed, users can see the Altera PCI API Driver under the device manager window, as shown in **Figure 5-9**.

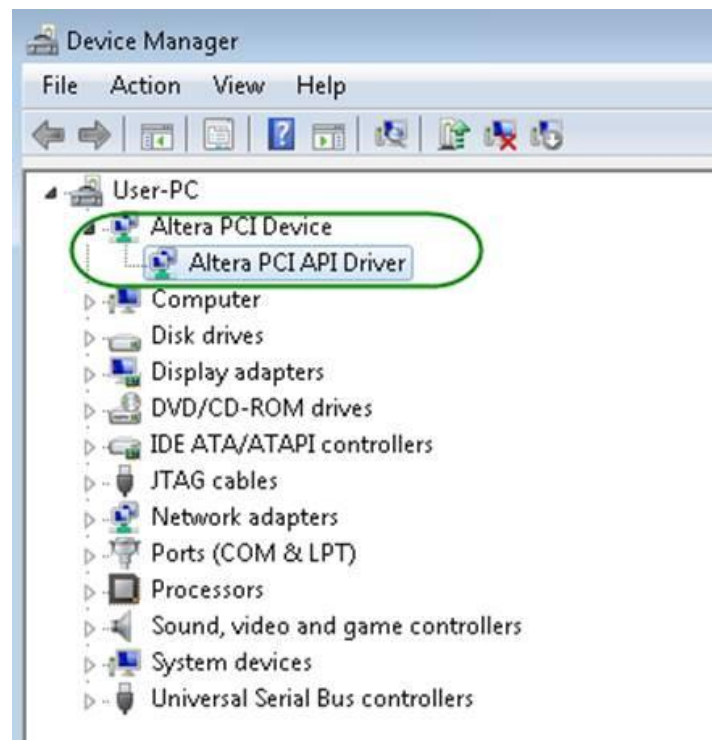


Figure 5-9 Altera PCI API Driver in Device Manager

■ Create a Software Application

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

5.4 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 longer used.

■ PCIe_Close

Function:
Close a handle associated to the PCIe card.
Prototype:
void PCIe_Close(PCI_HANDLE hPCI);
Parameters:
hPCI: 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(PCI_HANDLE hPCI, PCI_BAR PcieBar, PCI_ADDRESS PcieAddress, uint32_t *pdwData);
Parameters:
hPCI: 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(PCI_HANDLE hPCI,

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.

5.5 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. Set MSEL[2:0] to 010.
2. Install the FPGA board on your PC as shown in **Figure 5-10**.

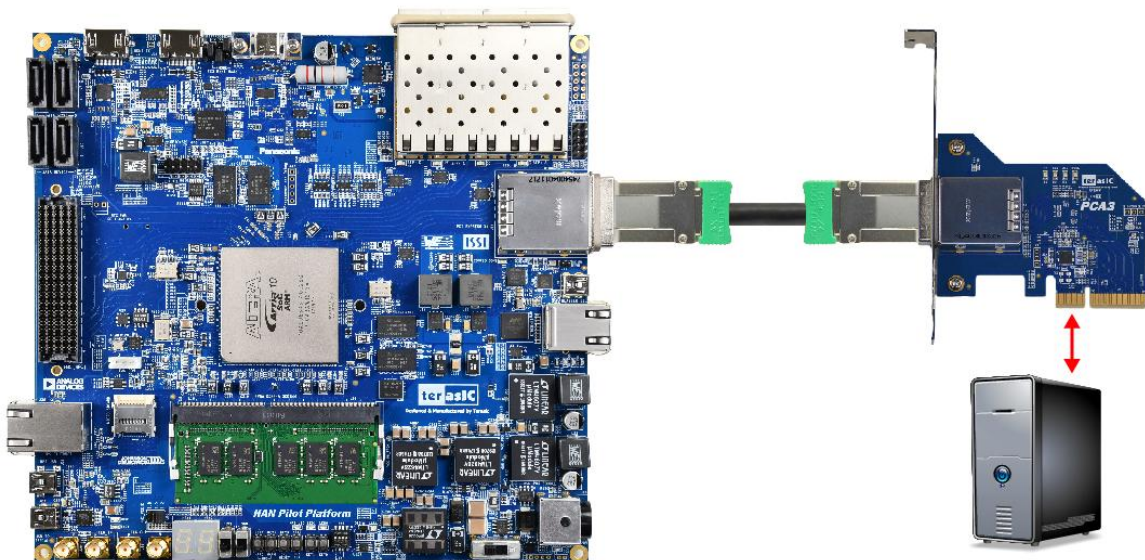


Figure 5-10 FPGA board connect to PC

3. Configure FPGA with PCIe_Fundamental.sof by executing the test.bat.
4. Install PCIe driver if necessary. The driver is located in the folder: CDROM\Demonstration\PCIe_SW_KIT\Windows\PCIe_Driver.
5. Restart Windows
6. Make sure the Windows has detected the FPGA Board by checking the Windows Control panel as shown in **Figure 5-11**.

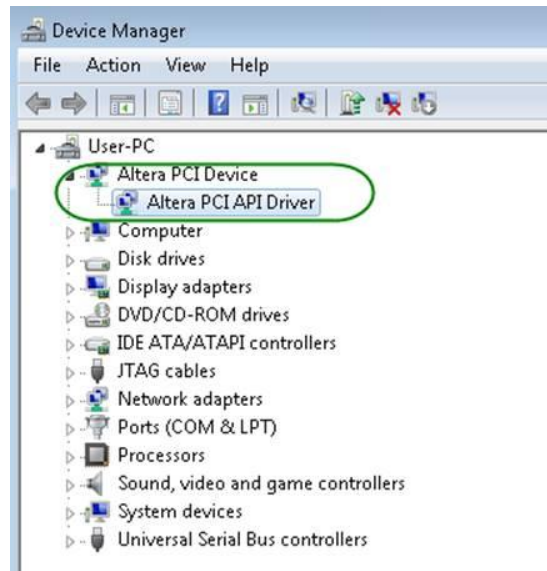


Figure 5-11 Screenshot for PCIe Driver

7. Goto windows_app folder, execute PCIE_FUNDAMENTAL.exe. A menu will appear as shown in **Figure 5-12**.

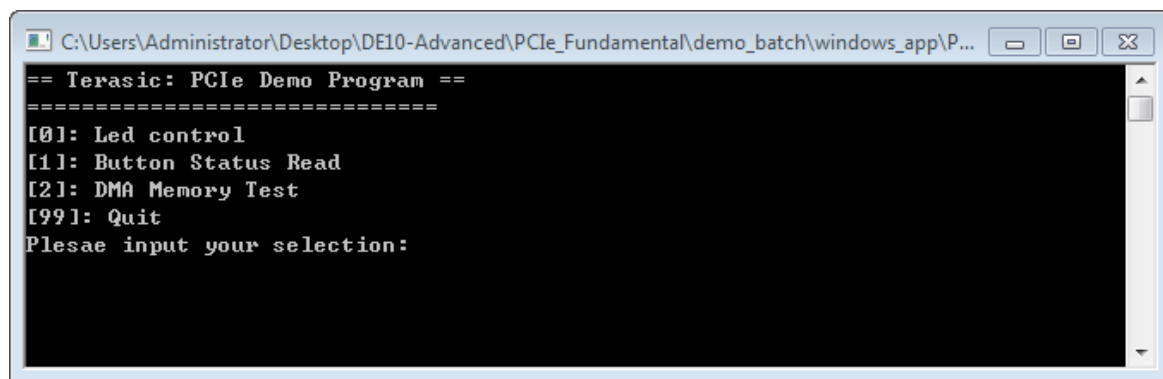
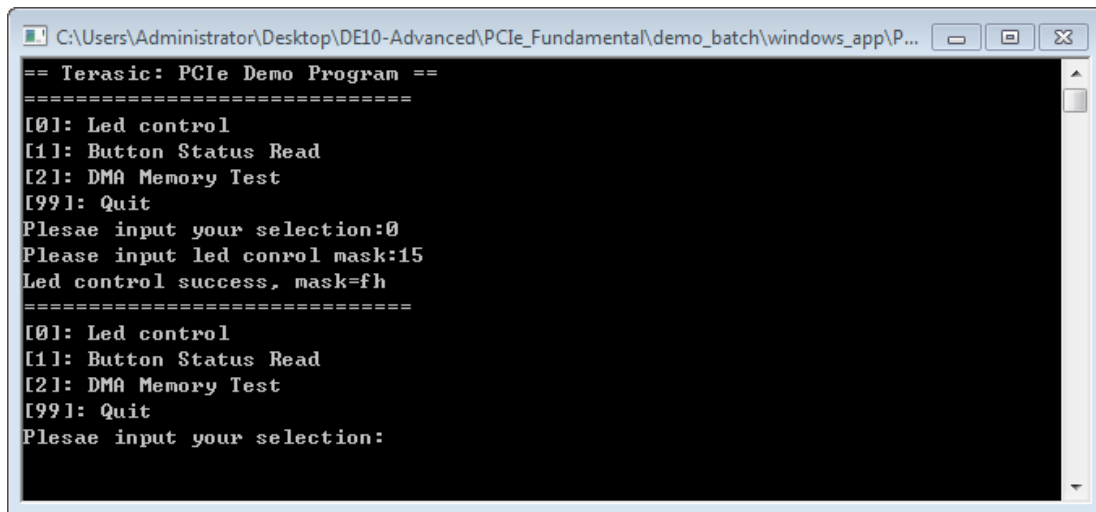


Figure 5-12 Screenshot of Program Menu

8. 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 5-13**. If input 0 (hex 0x00), all led will be turn off.

A screenshot of a Windows command prompt window titled "C:\Users\Administrator\Desktop\DE10-Advanced\PCIe_Fundamental\demo_batch\windows_app\P...". The window displays the "Terasic: PCIe Demo Program" menu with options: [0]: Led control, [1]: Button Status Read, [2]: DMA Memory Test, and [99]: Quit. The user has entered '0' for LED control, and the program has responded with "Led control success, mask=fh". The menu is shown again, and the user is prompted to enter a selection.

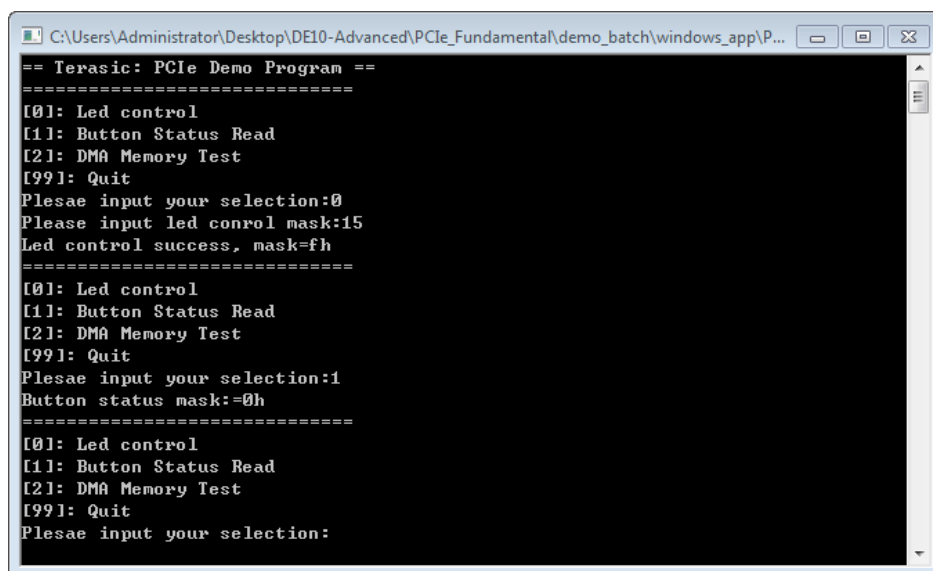
```
== Terasic: PCIe Demo Program ==  
=====
```

```
[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 5-13 Screenshot of LED Control

9. Type 1 followed by an ENTER key to select Button Status Read item. The button status will be report as shown in **Figure 5-14**.

A screenshot of the same Windows command prompt window. The user has entered '1' for Button Status Read, and the program has responded with "Button status mask:=0h". The menu is shown again, and the user is prompted to enter a selection.

```
== Terasic: PCIe Demo Program ==  
=====
```

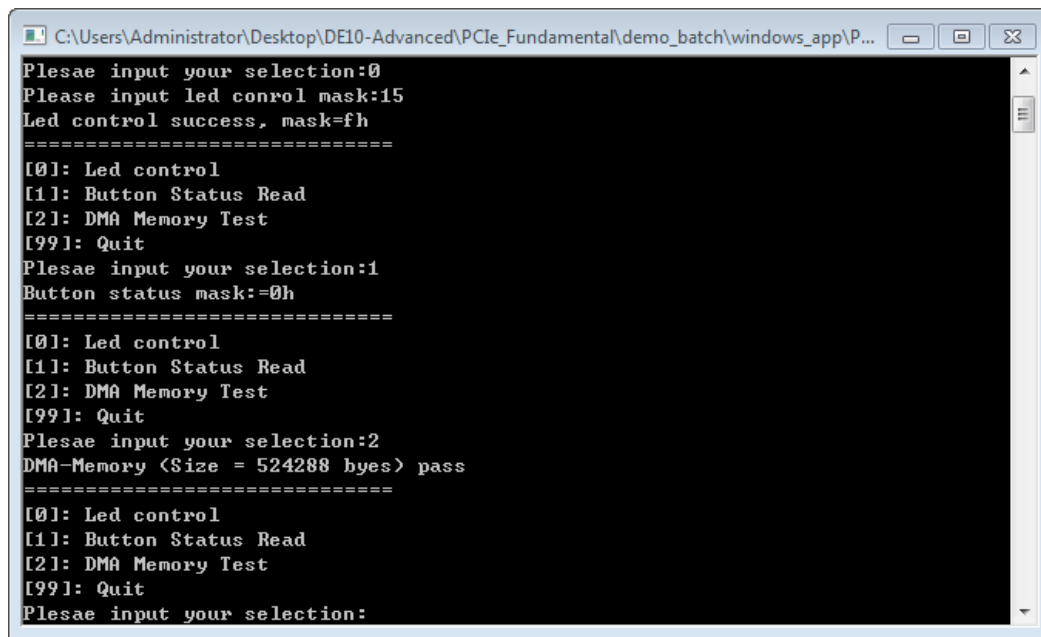
```
[0]: Led control  
[1]: Button Status Read  
[2]: DMA Memory Test  
[99]: Quit  
Plesae input your selection:0  
Please input led conrol mask:15  
Led control success, mask=fh  
=====
```

```
[0]: Led control  
[1]: Button Status Read  
[2]: DMA Memory Test  
[99]: Quit  
Plesae input your selection:1  
Button status mask:=0h  
=====
```

```
[0]: Led control  
[1]: Button Status Read  
[2]: DMA Memory Test  
[99]: Quit  
Plesae input your selection:
```

Figure 5-14 Screenshot of Button Status Report

10. Type 2 followed by an ENTER key to select DMA Testing item. The DMA test result will be report as shown in **Figure 5-15**.



```
C:\Users\Administrator\Desktop\DE10-Advanced\PCIE_Fundamental\demo_batch\windows_app\P...
Plesae input your selection:0
Please input led control 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:2
DMA-Memory <Size = 524288 bytes> pass
=====
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 5-15 Screenshot of DMA Memory Test Result

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

■ Development Tools

- Quartus Prime 18.0 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 5-16 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.

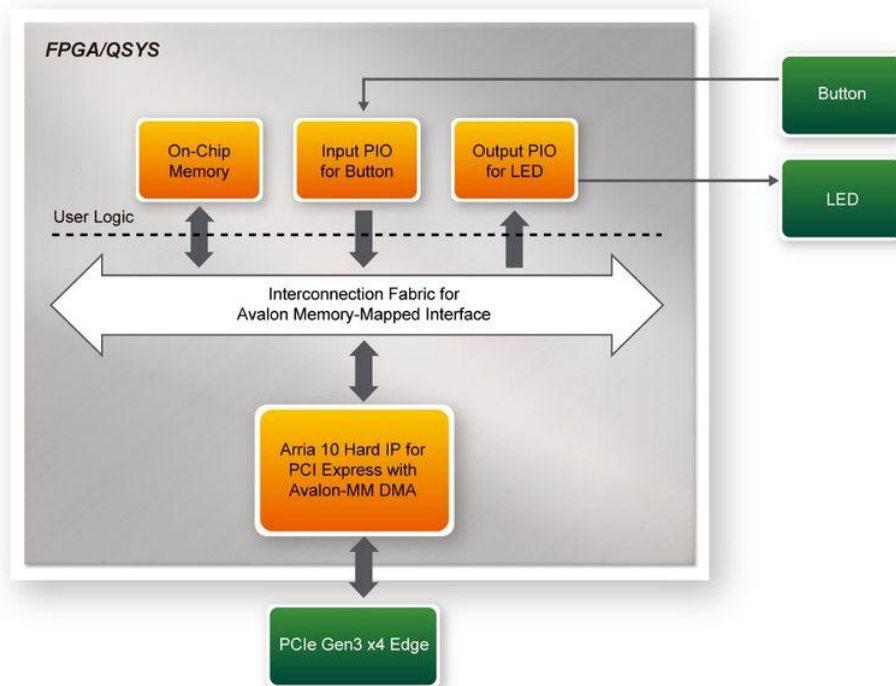


Figure 5-16 Hardware block diagram of the PCIe reference design

■ Windows Based Application Software Design

The application software project is built by Visual C++ 2012. The project includes the following major files as listed in [Table 5-1](#).

Table 5-1 Project 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, as shown in [Figure 5-17](#).

```
#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
```

Figure 5-17 Header file "PCIE.h"

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 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 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);
```

5.6 PCIe Reference Design - DDR4

The application reference design shows how to add DDR4 Memory Controllers for DDR4-A SODIMM and on board DDR4-B into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 4GB data DMA for both SODIMM. Also, this demo shows how to call “PCIE_ConfigRead32” API to check PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\Demonstrations\PCIe_DDR4\demo_batch

The folder includes following files:

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

- Terasic_PCIE_AVMM.dll

■ Demonstration Setup

1. Install DDR4 2400 4GB SODIMM on the FPGA board.
2. Set MSEL[2:0] to 010.
3. Install the FPGA board on your PC as shown in **Figure 5-18**.

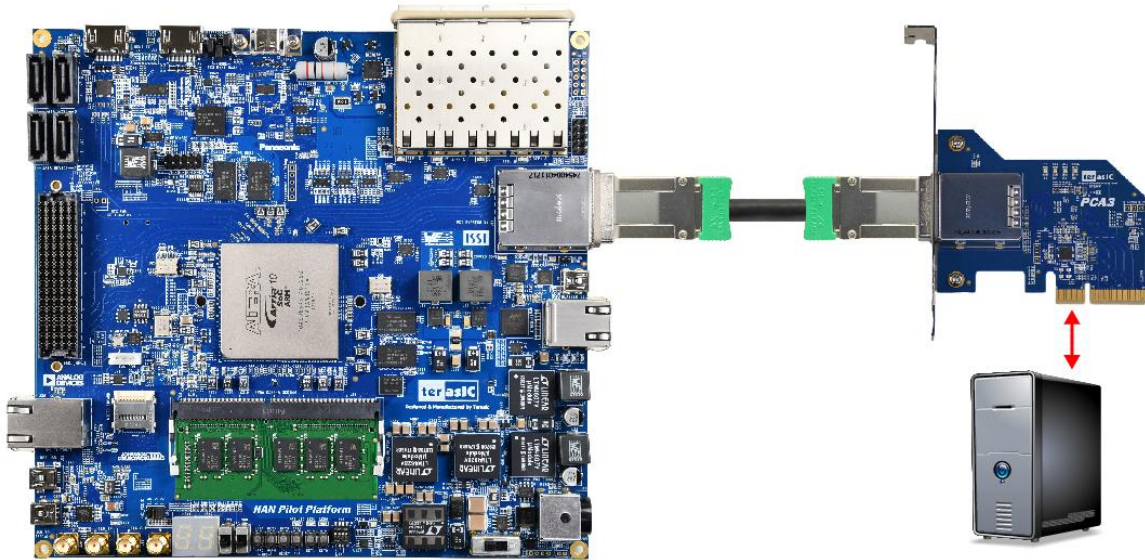


Figure 5-18 FPGA board connect to PC

4. Configure FPGA with PCIe_DDR4.sof by executing the test.bat.
5. Install PCIe driver if necessary.
6. Restart Windows
7. Make sure the Windows has detected the FPGA Board by checking the Windows Control panel.
8. Goto windows_app folder, execute PCIe_DDR4.exe. A menu will appear as shown in **Figure 5-19**.
9. Type 2 followed by a ENTER key to select Link Info item. The PCIe link information will be shown as in **Figure 5-20**. Gen3 link speed and x8 link width are expected.
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 5-21**.

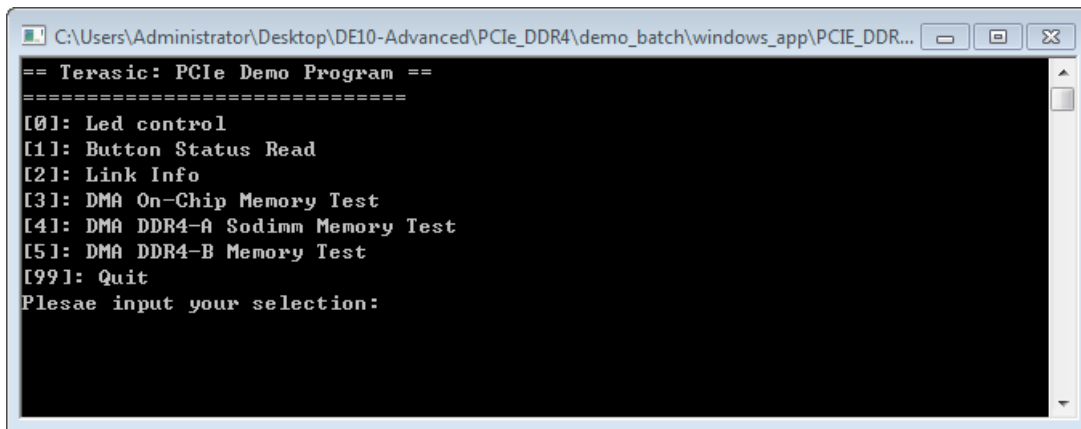


Figure 5-19 Screenshot of Program Menu

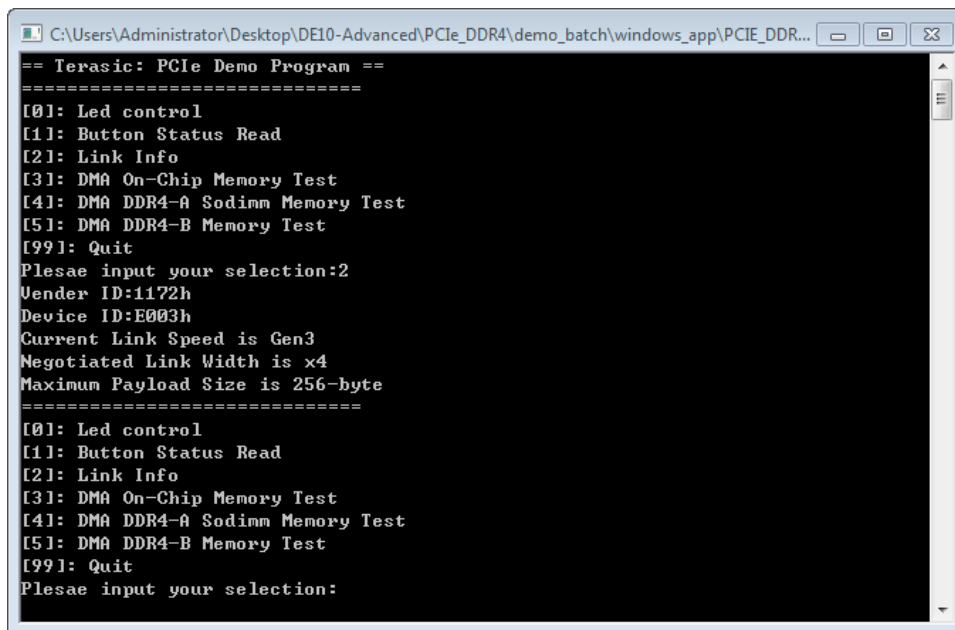


Figure 5-20 Screenshot of Link Info

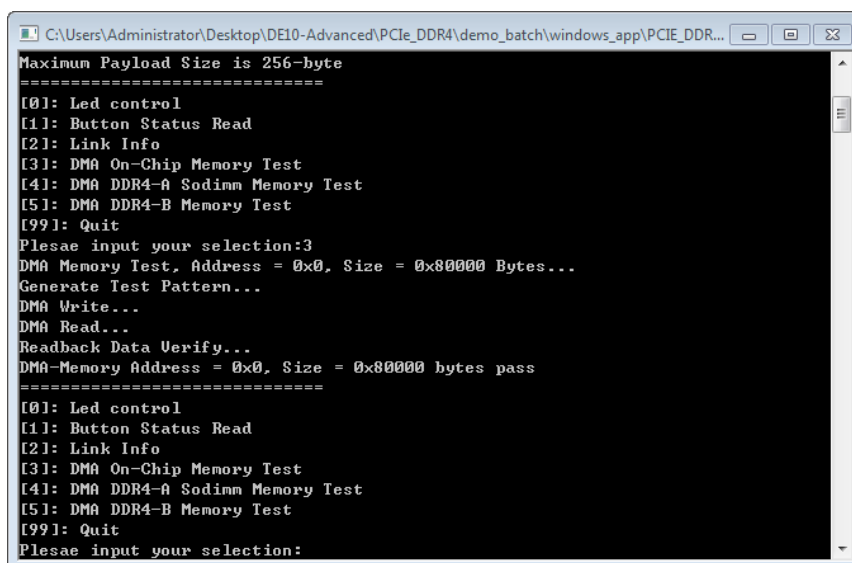
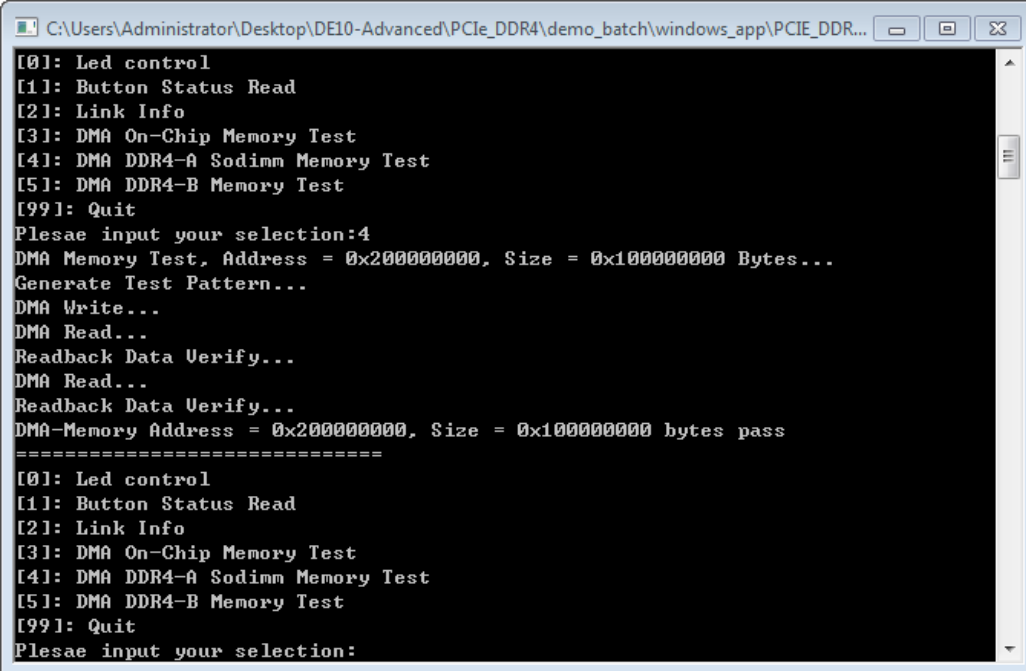


Figure 5-21 Screenshot of On-Chip Memory DMA Test Result

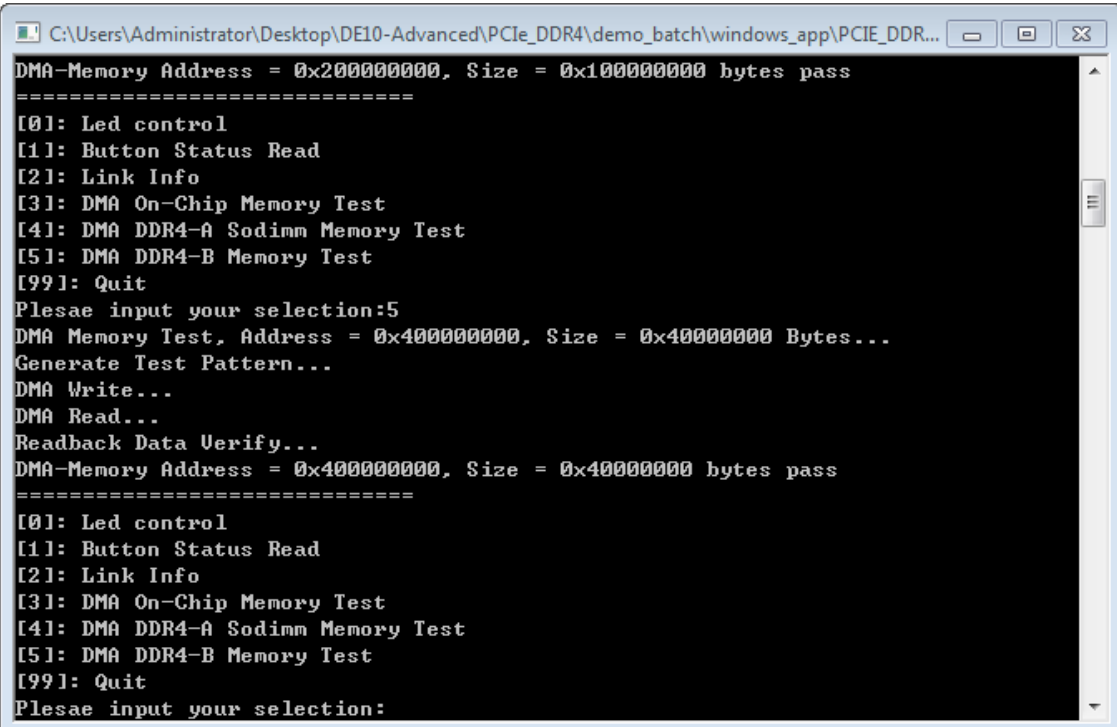
11. Type 4 followed by an ENTER key to select DMA DDR4-A SODIMM Memory Test item. The DMA write and read test result will be report as shown in [Figure 5-22](#).



```
C:\Users\Administrator\Desktop\DE10-Advanced\PCIE_DDR4\demo_batch\windows_app\PCIE_DDR...
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:4
DMA Memory Test, Address = 0x200000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x200000000, Size = 0x100000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 5-22 Screenshot of DDR4-A SOSIMM Memory DAM Test Result

12. Type 5 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 5-23](#).



```
C:\Users\Administrator\Desktop\DE10-Advanced\PCIE_DDR4\demo_batch\windows_app\PCIE_DDR...
DMA-Memory Address = 0x200000000, Size = 0x100000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:5
DMA Memory Test, Address = 0x400000000, Size = 0x400000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x400000000, Size = 0x400000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 5-23 Screenshot of DDR4-B SOSIMM Memory DAM Test Result

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

■ Development Tools

- Quartus Prime 18.0 Standard Edition
- Visual C++ 2012

■ Demonstration Source Code Location

- Quartus Project: Demonstrations\PCIE_DDR4
- Visual C++ Project: Demonstrations\PCIE_SW_KIT\Windows\PCIE_DDR4

■ FPGA Application Design

Figure 5-24 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.

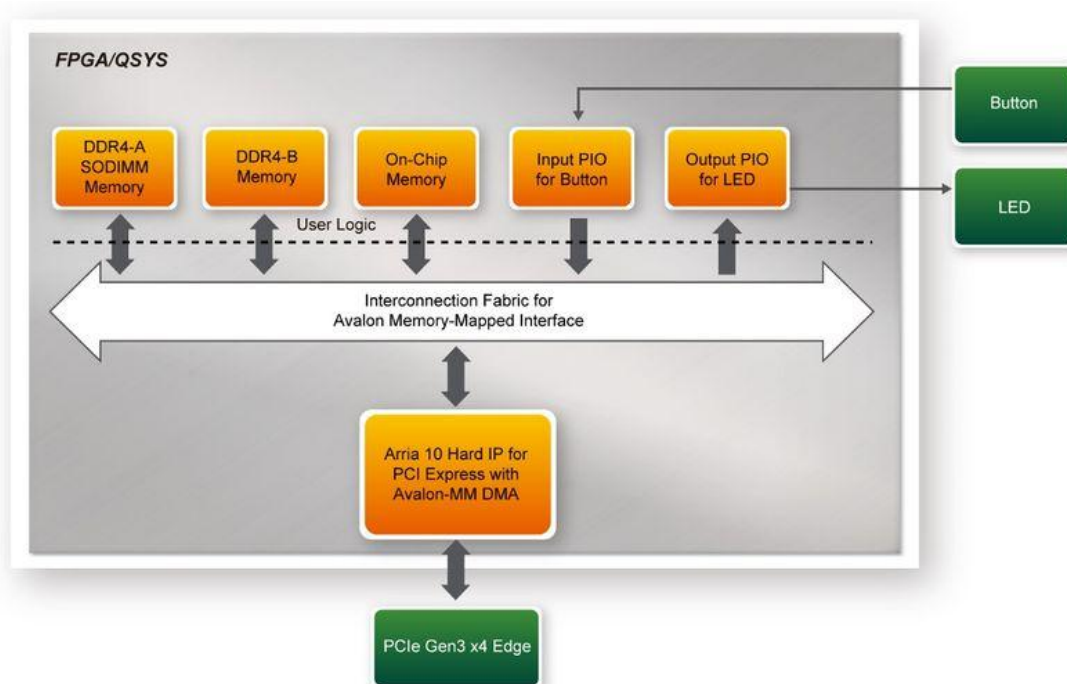


Figure 5-24 Hardware block diagram of the PCIe_DDR4 reference design

■ 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_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_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_IO_BUTTON_ADDR    0x4000020
#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         (1ull*1024*1024*1024) //1GB
```

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;
}

```

PCI Express 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](#)

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

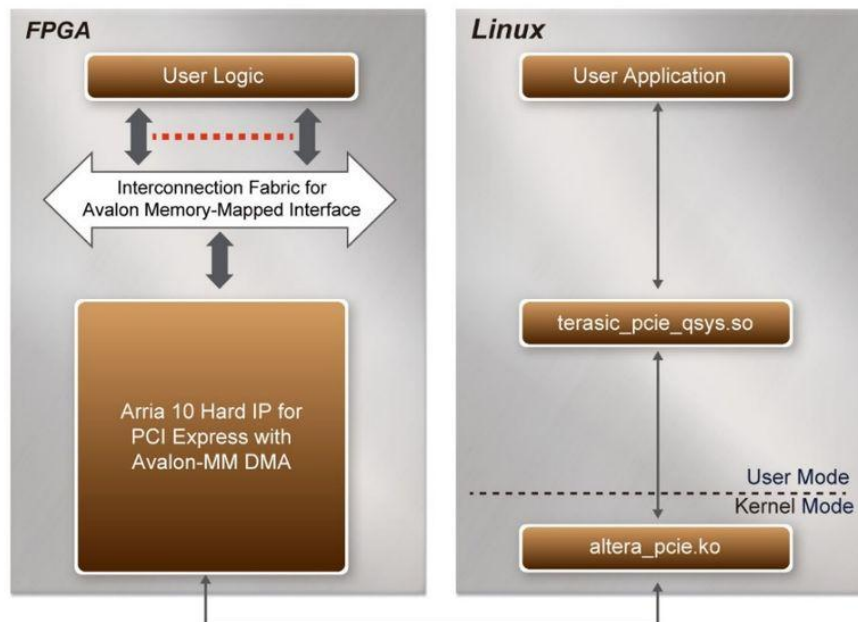


Figure 6-1 Infrastructure of PCI Express System

6.2 PC PCI Express Software SDK

The FPGA System CD contains a PC Windows based SDK to allow users to develop their 64-bit software application on 64-bits Linux. CentOS 7.2 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.

6.3 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_qsys.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.

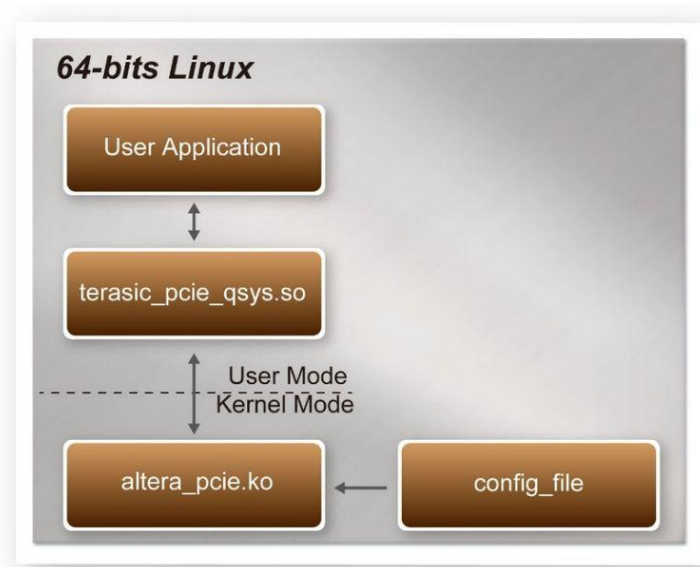


Figure 6-2 PCI Express Software Stack

■ Install PCI Express Driver on Linux

To make sure the PCIe driver can meet your kernel of Linux distribution, the driver `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. Make sure the HAN Pilot Platform and the PC are both powered off.
2. Set MSEL[2:0] to 010.
3. Plug the PCIe adapter card into the PCIe slot on the PC motherboard. Use the PCIe cable to connect to the HAN Pilot Platform PCIe connector and the PCIe adapter card (See **Figure 6-3**)

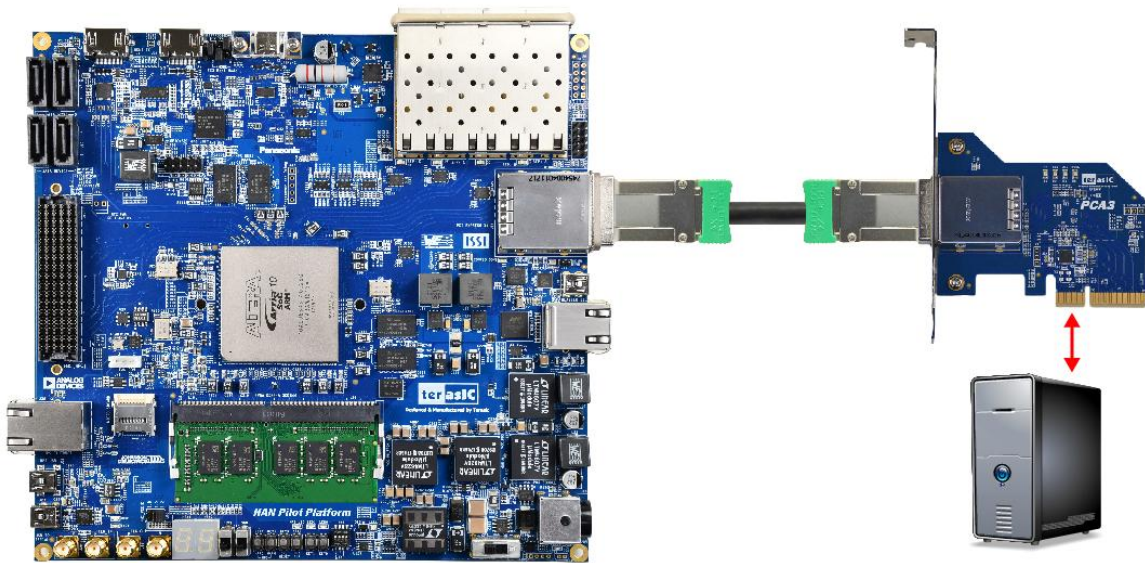


Figure 6-3 FPGA board connect to PC

4. Power on your HAN Pilot Platform and the host PC.
5. Open a terminal and use "cd" command to goto the folder "CDROM/ Demonstrations/ PCIe_Fundamental/demo_batch".
6. Set `QUARTUS_ROOTDIR` variable pointing to the Quartus installation path. Set `QUARTUS_ROOTDIR` variable by tying the following commands in terminal. Replace "/home/centos/intelFPGA/18.0/quartus" to your quartus installation path.


```
export QUARTUS_ROOTDIR=/home/centos/intelFPGA/18.0/quartus
```

7. Execute "sudo -E sh test.sh" command to configure the FPGA
8. Restart Linux operation system. In Linux, open a terminal and use "cd" command to goto the PCIe_Driver folder.
9. 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 6-4**.

```
make
```

```
sudo sh load_driver
```

```
dmesg | tail -n 15
```

```
[root@localhost PCIe_Driver]# sudo sh load_driver
Matching Device Found
[root@localhost PCIe_Driver]# dmesg | tail -n 15
[ 524.796078] SELinux: initialized (dev tmpfs, type tmpfs), uses transition SIDs
[ 719.904919] Altera PCIe: altera_pcie_init(), Aug 15 2018 17:13:37
[ 719.905017] Altera PCIe 0000:01:00.0: pci_enable_device() successful
[ 719.905049] Altera PCIe 0000:01:00.0: irq 31 for MSI/MSI-X
[ 719.905060] Altera PCIe 0000:01:00.0: pci_enable_msi() successful
[ 719.905065] Altera PCIe 0000:01:00.0: BAR[0] 0xe8000000-0xe80001ff flags 0x0014220c, length 512
[ 719.905067] Altera PCIe 0000:01:00.0: BAR[1] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 719.905070] Altera PCIe 0000:01:00.0: BAR[2] 0xf7c01000-0xf7c0100f flags 0x00040200, length 16
[ 719.905072] Altera PCIe 0000:01:00.0: BAR[3] 0xf7c00000-0xf7c0000f flags 0x00040200, length 16
[ 719.905075] Altera PCIe 0000:01:00.0: BAR[4] 0xe0000000-0xe7ffffff flags 0x0014220c, length 134217728
[ 719.905077] Altera PCIe 0000:01:00.0: BAR[5] 0x00000000-0x00000000 flags 0x00000000, length 0
[ 719.905093] Altera PCIe 0000:01:00.0: BAR[0] mapped to 0xffffc900074b6000, length 512
[ 719.905099] Altera PCIe 0000:01:00.0: BAR[2] mapped to 0xffffc900074f4000, length 16
[ 719.905104] Altera PCIe 0000:01:00.0: BAR[3] mapped to 0xffffc900074f6000, length 16
[ 719.905347] Altera PCIe 0000:01:00.0: BAR[4] mapped to 0xffffc9001c180000, length 134217728
[root@localhost PCIe_Driver]#
```

Figure 6-4 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 list 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 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.

6.4 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 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: <pre>bool PCIE_Read8(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, uint8_t *pByte);</pre>
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: <pre>bool PCIE_Write8(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, uint8_t Byte);</pre>
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: <pre>bool PCIE_DmaRead(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pBuffer, uint32_t dwBufSize);</pre>
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: <pre>bool PCIE_DmaWrite(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, uint32_t dwDataSize);</pre>
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.

6.5 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. Set MSEL[2:0] to 010.
2. Install the FPGA board on your PC as shown in **Figure 6-5**.

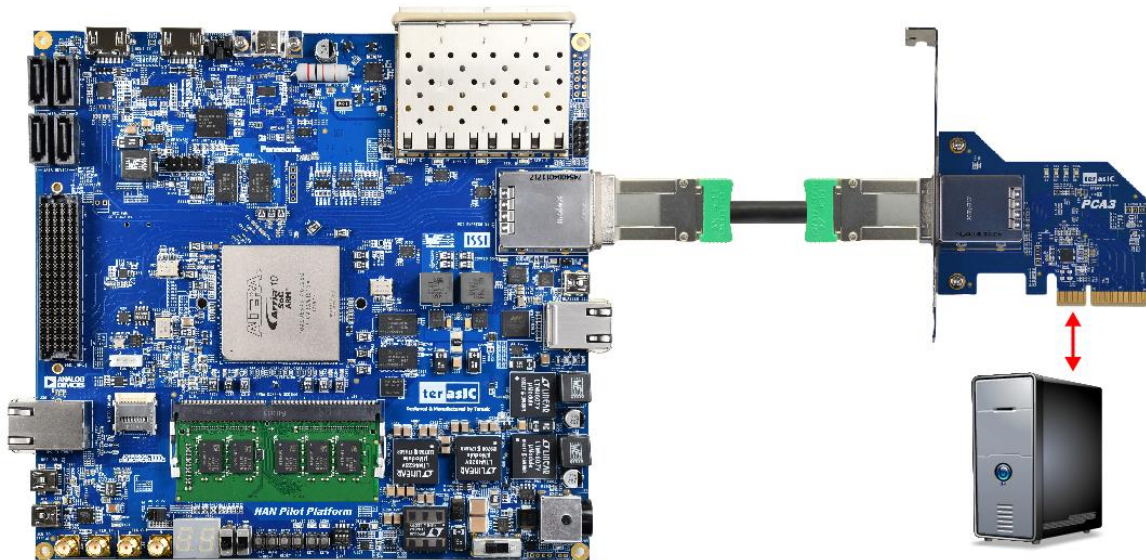


Figure 6-5 FPGA board connect to PC

3. Open a terminal and use "cd" command to go to:
CDROM/ Demonstrations/ PCIe_Fundamental/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/centos/intelFPGA/18.0/quartus to your quartus installation path.

export QUARTUS_ROOTDIR=/home/centos/intelFPGA/18.0/quartus
5. Execute "sudo -E sh test.sh" command to configure the FPGA.
6. Restart Linux.
7. Install PCIe driver. The driver is located in the folder:
CDROM/Demonstration/PCIe_SW_KIT/Linux/PCIe_Driver.
8. 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 in **Figure 6-6**.

```
[root@localhost Desktop]# ls -l /dev/altera_pcie*
crw-rw-rw-. 1 root wheel 245, 0 Aug 15 17:13 /dev/altera_pcie0
[root@localhost Desktop]#
```

Figure 6-6 Linux has detected the FPGA Board

9. Go to linux_app folder, execute PCIE_FUNDAMENTAL. A menu will appear as shown in **Figure 6-7**.

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_FUNDAMENTAL/demo_batch/linux_app - □ ×
[root@localhost 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:█
```

Figure 6-7 Screenshot of Program Menu

10. 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 6-8**. If input 0 (hex 0x00), all led will be turn off.

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_FUNDAMENTAL/demo_batch/linux_app - □ ×
[root@localhost 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 control mask:3
Led control success, mask=3h
=====
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:█
```

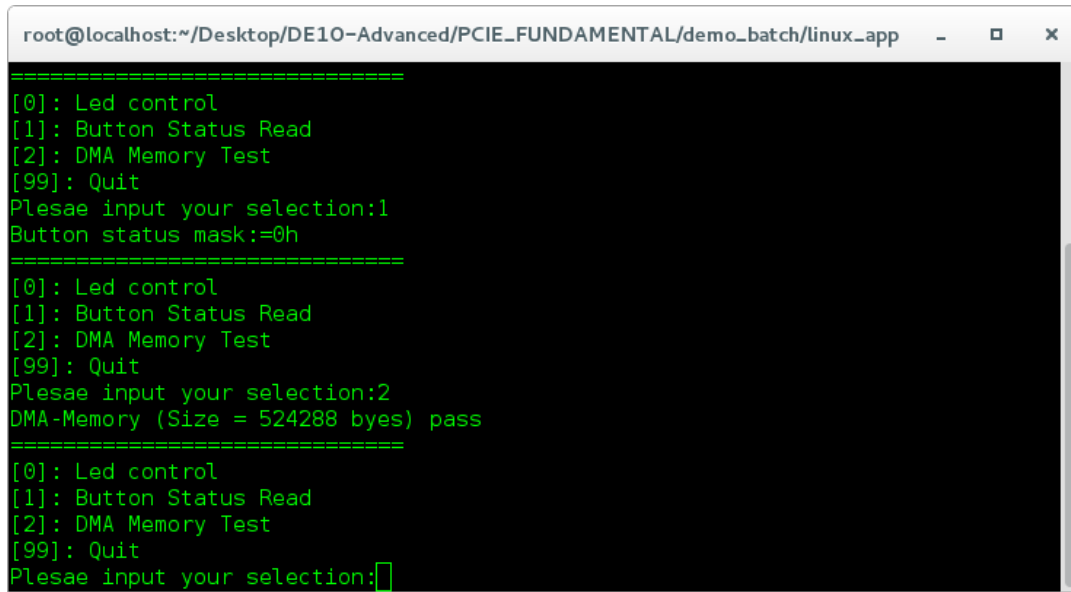
Figure 6-8 Screenshot of LED Control

11. Type 1 followed by an ENTER key to select Button Status Read item. The button status will be report as shown in **Figure 6-9**.

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_FUNDAMENTAL/demo_batch/linux_app - □ ×
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:0
Please input led control mask:3
Led control success, mask=3h
=====
[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 6-9 Screenshot of Button Status Report

12. Type 2 followed by an ENTER key to select DMA Testing item. The DMA test result will be report as shown in **Figure 6-10**.



```
root@localhost:~/Desktop/DE10-Advanced/PCIE_FUNDAMENTAL/demo_batch/linux_app - □ x
=====
[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:2
DMA-Memory (Size = 524288 bytes) pass
=====
[0]: Led control
[1]: Button Status Read
[2]: DMA Memory Test
[99]: Quit
Plesae input your selection:█
```

Figure 6-10 Screenshot of DMA Memory Test Result

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

■ Development Tools

- Quartus Prime 18.0 Standard Edition
- GNU Compiler Collection, Version 4.8 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstrations/PCIE_Fundamental
- C++ Project: Demonstrations/PCIE_SW_KIT/Linux/PCIE_FUNDAMENTAL

■ FPGA Application Design

Figure 6-11 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.

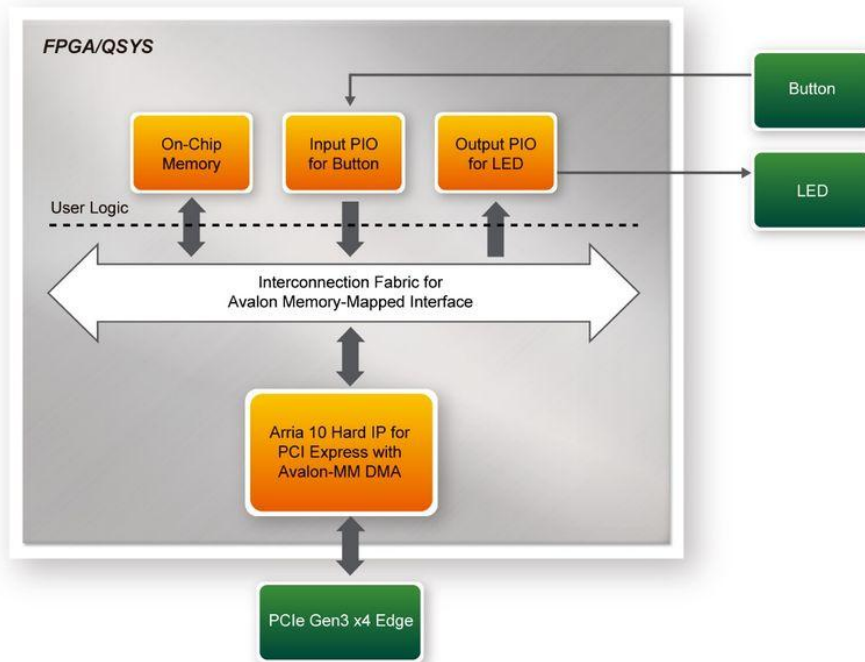


Figure 6-11 Hardware block diagram of the PCIe reference design

■ Linux Based Application Software Design

The application software project is built by GNU Toolchain. The project includes the following major files as shown in **Table 6-1**.

Table 6-1 Project 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, as shown in **Figure 6-12**.

```
#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
```

Figure 6-12 Header file "PCIE.h"

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 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 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);
```

6.6 PCIe Reference Design - DDR4

The application reference design shows how to add DDR4 Memory Controllers for DDR4-A SODIMM and on board DDR4-B into the PCIe Quartus project based on the PCIe_Fundamental Quartus project and perform 4GB data DMA for both SODIMM. Also, this demo shows how to call “PCIE_ConfigRead32” API to check PCIe link status.

■ Demonstration Files Location

The demo file is located in the batch folder:

CDROM\ Demonstrations\PCIe_DDR4\demo_batch

The folder includes following files:

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

- terasic_pcie_qsys.so

■ Demonstration Setup

1. Install DDR4 2400 4GB SODIMM on the FPGA board.
2. Set MSEL[2:0] to 010.
3. Install the FPGA board on your PC as shown in **Figure 6-13**.

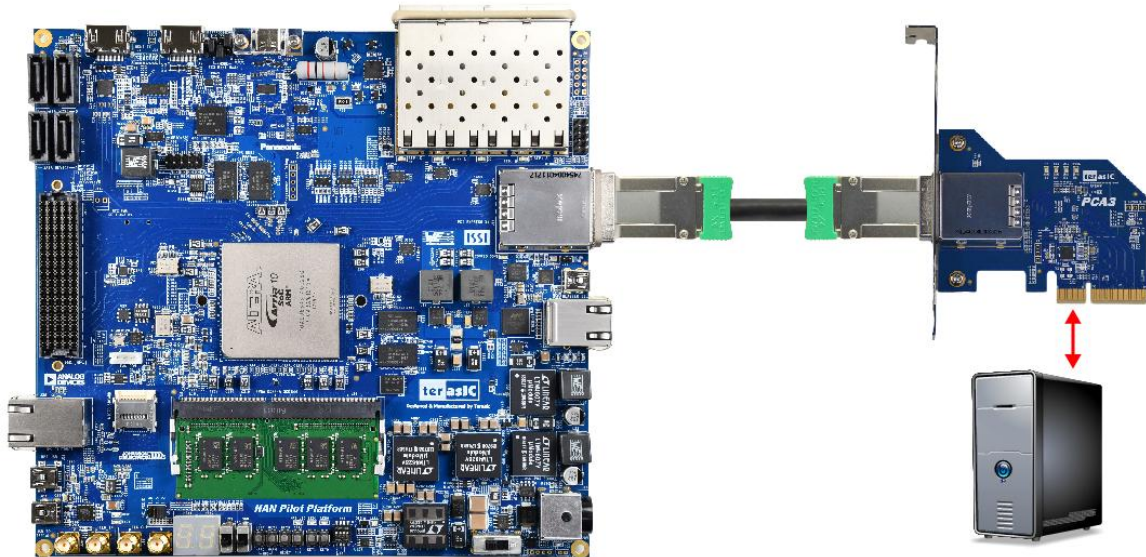


Figure 6-13 FPGA board connect to PC

4. Open a terminal and use "cd" command to go to "CDROM/Demonstrations/PCie_Fundamental/demo_batch".
5. Set QUARTUS_ROOTDIR variable pointing to the Quartus installation path. Set QUARTUS_ROOTDIR variable by typing the following commands in terminal. Replace /home/centos/intelFPGA/18.0/quartus to your quartus installation path.

```
export QUARTUS_ROOTDIR=/home/centos/intelFPGA/18.0/quartus
```
6. Execute "sudo -E sh test.sh" command to configure the FPGA
7. Restart Linux.
8. Install PCIe driver.
9. Make sure the Linux has detected the FPGA Board.
10. Go to linux_app folder, execute PCIE_DDR4. A menu will appear as shown in **Figure 6-14**.


```
root@localhost:~/Desktop/DE10-Advanced/PCIE_DDR4/demo_batch/linux_app - □ x
[root@localhost linux_app]# ./PCIE_DDR4
== Terasic: PCie Demo Program ==
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:[]
```

Figure 6-14 Screenshot of Program Menu

11. Type 2 followed by an ENTER key to select Link Info item. The PCIe link information will be shown as in [Figure 6-15](#). Gen3 link speed and x8 link width are expected.

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_DDR4/demo_batch/linux_app - □ x
[root@localhost linux_app]# ./PCIE_DDR4
== Terasic: PCie Demo Program ==
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:2
Vender ID:1172h
Device ID:E003h
Current Link Speed is Gen3
Negotiated Link Width is x4
Maximum Payload Size is 256-byte
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:█
```

Figure 6-15 Screenshot of Link Info

12. 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 6-16](#).

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_DDR4/demo_batch/linux_app - □ x
Maximum Payload Size is 256-byte
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[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 DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:█
```

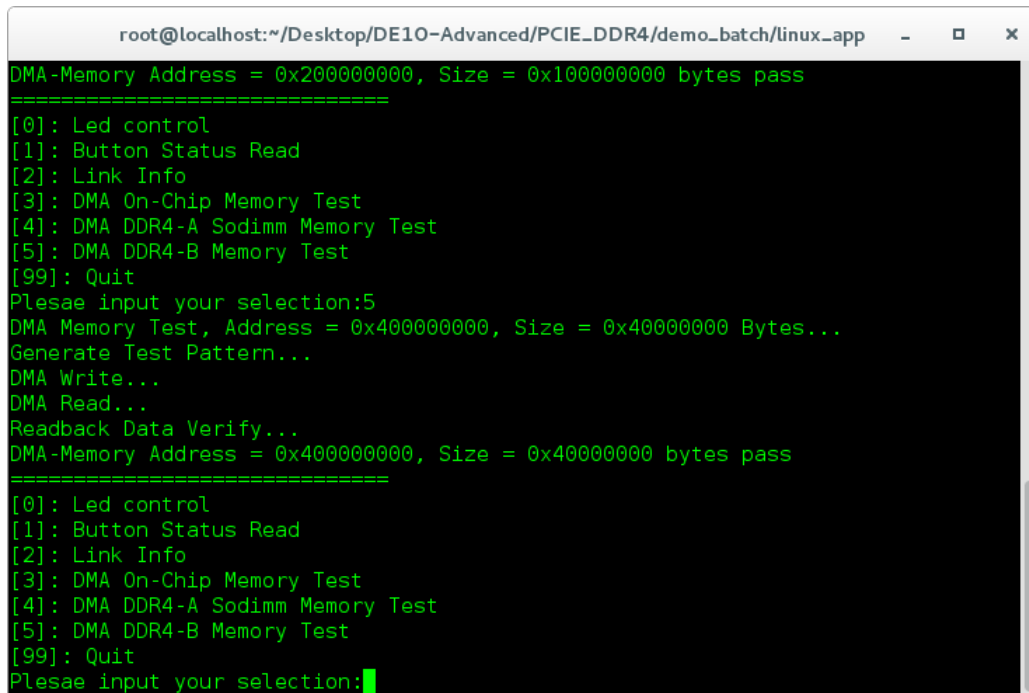
Figure 6-16 Screenshot of On-Chip Memory DMA Test Result

13. Type 4 followed by an ENTER key to select DMA DDR4-A SODIMM Memory Test item. The DMA write and read test result will be report as shown in **Figure 6-17**.

```
root@localhost:~/Desktop/DE10-Advanced/PCIE_DDR4/demo_batch/linux_app - □ x
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:4
DMA Memory Test, Address = 0x200000000, Size = 0x100000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x200000000, Size = 0x100000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:█
```

Figure 6-17 Screenshot of DDR4-A SOSIMM Memory DAM Test Result

14. Type 5 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 6-18](#).



```
root@localhost: ~/Desktop/DE10-Advanced/PCIE_DDR4/demo_batch/linux_app
DMA-Memory Address = 0x200000000, Size = 0x100000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:5
DMA Memory Test, Address = 0x400000000, Size = 0x40000000 Bytes...
Generate Test Pattern...
DMA Write...
DMA Read...
Readback Data Verify...
DMA-Memory Address = 0x400000000, Size = 0x40000000 bytes pass
=====
[0]: Led control
[1]: Button Status Read
[2]: Link Info
[3]: DMA On-Chip Memory Test
[4]: DMA DDR4-A Sodimm Memory Test
[5]: DMA DDR4-B Memory Test
[99]: Quit
Plesae input your selection:
```

Figure 6-18 Screenshot of DDR4-B SODIMM Memory DAM Test Result

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

■ Development Tools

- Quartus Prime 18.0 Standard Edition
- GNU Compiler Collection, Version 4.8 is recommended

■ Demonstration Source Code Location

- Quartus Project: Demonstrations\PCIE_DDR4
- Visual C++ Project: Demonstrations\PCIE_SW_KIT\Windows\PCIE_DDR4

■ FPGA Application Design

Figure 6-19 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.

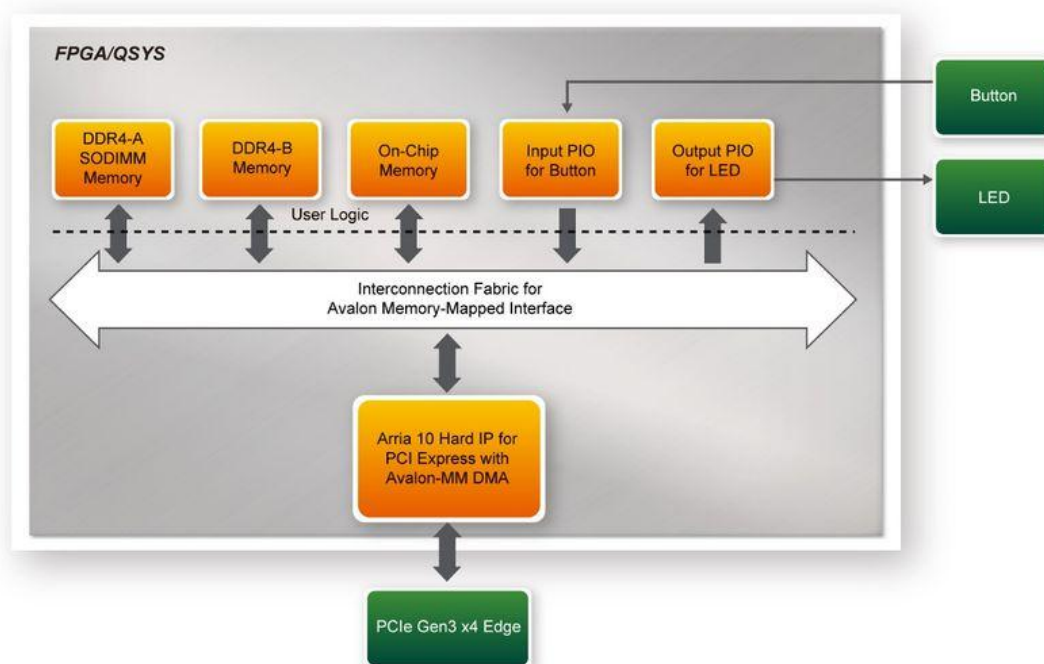


Figure 6-19 Hardware block diagram of the PCIe_DDR4 reference design

■ Linux 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 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_IO_BUTTON_ADDR    0x4000020
#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          (1ull*1024*1024*1024) //1GB
```

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;
}
```

7.1 Introduction

The HAN Pilot Platform Kit includes Linux BSP (Board Support Package) with which users can develop their software application on the Linux. The Kit contains the three Linux BSP listed below. Users can select the proper BSP for their applications.

- Linux LXDE VNC Desktop BSP
- Linux LXDE HDMI Desktop BSP
- VNC Desktop OpenCL BSP

Please be aware that not all WiFi/Camera/Audio USB dongles are compatible with this BSP. Here are some compatible USB dongles that have been tested, by us, and proven to work. The following suggested WiFi USB dongles can be purchased from the Terasic Website.

- WiFi USB Dongle
 - Mi WiFi (Terasic PN: FXX-3061-MIX)
- Camera USB Dongle
 - Logitech C310
 - ET USB 2760 Camera
 - Genius WideCam F100
- Audio USB Dongle
 - Ugreen US205

These Linux BSP can be downloaded for free from the Terasic Website:

[http:// HAN Pilot Platform.terasic.com/cd](http://HANPilotPlatform.terasic.com/cd)

7.2 Use Linux BSP

This section describes the procedure to boot Linux on HAN Pilot Platform. For more details, refer to chapter 5 of **HAN Pilot Platform_Getting_Started_Guide.pdf** in the System CD.

- Download the BSP image file from [http:// HAN Pilot Platform.terasic.com/cd](http://HANPilotPlatform.terasic.com/cd)
- Create a Linux booting microSD card by using Win32 Disk Imager utility to write the image file into a microSD card
- Insert the microSD to microSD socket on the HAN Pilot Platform
- Make sure MSEL[2:0] switch on HAN Pilot Platform is set to proper position
- For VNC mode:
 - Connect your host PC to USB-to-Serial port(J27) on HAN Pilot Platform via a USB cable
 - Launch the Putty software in your PC
 - The booting message will appear on the Putty terminal
 - In host PC, use vnc client to connect to HAN Pilot Platform

- For HDMI mode:
 - Connect a HDMI monitor, an USB keyboard, and a USB mouse to the HAN Pilot Platform with USB Hub
 - Power on HAN Pilot Platform
 - The LXDE Desktop will appear on the HDMI monitor

7.3 Linux LXDE VNC Desktop BSP

This is a VNC mode Linux BSP. The console content is displayed on the UART Terminal in your Host PC and the Desktop is displayed on VNC Client. Refer to **Table 7-1** for Linux LXDE VNC Desktop BSP information.

Table 7-1 Linux LXDE VNC Desktop BSP Information

Item	Description
BSP Location	HAN Pilot Platform_VNC_Desktop.zip Download link: http:// HAN Pilot Platform.terasic.com/cd
MicroSD Card	4GB at minimal
MSEL[2:0]	000
Account	User name: root, password is not required (press Enter)
UART Terminal	Baud rate: 115200 Data bits: 8 Parity: None Stop Bits: 1 Flow Control: no
Quartus Project	a10s_ghrd
BSP Feature	USB Audio Dongle driver USB WiFi Dongle driver Example Codes
Linux Kernel Source	Source: https://github.com/terasic/linux-socfpga Branch: socfpga-4.5 Under above location: Configure File: HAN Pilot Platform_vnc.config DTS File: arch/arm/boot/dts/ HAN Pilot Platform_vnc.dts

7.4 Linux LXDE HDMI Desktop BSP

This is a Linux BSP with HDMI LXDE Desktop. The LXDE Desktop is displayed on the HDMI monitor attached to HAN Pilot Platform. **Table 7-2** describes the LXDE Desktop BSP items and lists the corresponding information. The BSP provides frame buffer for desktop display. The frame buffer function is implemented in FPGA site. The HPS ddr4 is used as video buffer in the frame buffer function.

Table 7-2 Linux LXDE HDMI Desktop BSP Information

Item	Description
BSP Location	HAN Pilot Platform_HDMI_Desktop.zip

	Download link: http:// HAN Pilot Platform.terasic.com/cd
MicroSD Card	4GB at minimal
MSEL[2:0]	000
Account	User name: root, password is not required (press Enter)
UART Terminal	Baud rate: 115200 Data bits: 8 Parity: None Stop Bits: 1 Flow Control: no
Quartus Project	Reserved
BSP Feature	LXDE Desktop Frame Buffer ALSA (Advanced Linux Sound Architecture) OpenCV Library GNU Toolchain USB WiFi Dongle driver and application example code USB Camera Dongle driver and OpenCV example code Example codes for accessing peripherals connected to FPGA and HPS.
Linux Kernel Source	Source: https://github.com/terasic/linux-socfpga Branch: socfpga-4.5 Configure File: Reserved DTS File: Reserved

7.5 VNC Desktop OpenCL BSP

This is a Linux BSP with VNC mode which supports Intel SDK OpenCL. The Intel® FPGA SDK for Open Computing Language (OpenCL™) allows a user to abstract away the traditional hardware FPGA development flow for a much faster and higher level software development flow. For more details, please refer to HAN Pilot Platform_OpenCL.pdf in the HAN Pilot Platform System CD. **Table 7-3** lists the OpenCL BSP component reference and the related information.

Table 7-3 OpenCL BSP Information

Item	Description
BSP Location	HAN Pilot Platform_OpenCL_BSP.zip Download link: http:// HAN Pilot Platform.terasic.com/cd
MicroSD Card	4GB at minimal
MSEL[2:0]	000
Account	User name: root, password is not required (press Enter)
UART Terminal	Baud rate: 115200 Data bits: 8 Parity: None Stop Bits: 1 Flow Control: no
Quartus Project	Reserved
BSP Feature	Frame Buffer

	OpenCL Example Codes
Linux Kernel Source	Source: https://github.com/terasic/linux-socfpga/tree/socfpga-3.10 Branch: socfpga-3.10 Under above location: Configure File: Reserved

Additional Information

Getting Help

Contact us via the following methods for further technical assistance:

Terasic Inc.

9F, No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, Taiwan 300-70

Email : support@terasic.com

Web : www.terasic.com

Revision History

Date	Version	Changes
2018.04.16	First publication	
2019.07.07	v1.1	Modify figures which are show as DE10-Advanced.
2019.09.06	v1.2	Modify default code section