

Apollo Agilex SoM Board

Demonstration manual



Copyright © Terasys Inc. All Rights Reserved.

FPGA

Contents

Chapter 1	Overview.....	4
Chapter 2	Examples For FPGA	5
2.1	Configure Si5340A in RTL.....	5
2.2	Basic Nios II control demo for SI5340A/ Temperature/ Power/ Fan	12
2.3	DDR4 SDRAM RTL Test	16
2.4	DDR4 SDRAM Test by Nios II	19
2.5	Board Information IP	24
2.6	100G Ethernet Example	28
2.7	24G CPRI Example.....	32
Chapter 3	Examples for HPS SoC.....	37
3.1	HPS 2x6 GPIO Header	37
3.2	HPS LED/KEY.....	41
3.3	Network Socket	44
3.4	Build C/C++ Project.....	51
Chapter 4	Additional Information	53
4.1	Getting Help	53



Chapter 1

Overview

This Manual will introduce the various application demonstrations on Apollo Agilex board. These demonstrations cover most of the interfaces on Apollo Agilex. Let users familiarize using these interfaces of the Apollo Agilex board. Demonstrations according to FPGA fabrics and HPS are divided into three categories:

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

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

- [Intel Quartus® Prime Pro Edition Software Version 21.4.0](#) or later.
- [Intel SoC Embedded Design Suite\(EDS\) Professional Edition](#)

Note: To run the demo bath file with the Nios II CPU of the demonstration on windows system, user need to install the Windows Subsystem for Linux (WSL) first then you can run the batch file. Please refer to the link to install : [Getting Start Install WSL](#)

Chapter 2

Examples For FPGA

This chapter provides examples of advanced designs implemented by RTL or Qsys on the Apollo Agilex SoM board. These reference designs cover the features of peripherals connected to the FPGA, such as DDR4, temperature monitor, PLL clock setting and Power monitor. All the associated files can be found in the directory \Demonstrations\FPGA of Apollo Agilex SoM System CD.

2.1 Configure Si5340A in RTL

There is a Silicon Labs Si5340A clock generator on Apollo Agilex board can provide adjustable frequency reference clock (See **Figure 2-1**) for FMC/FMC+ connectors and QSFP28 interface. The Si5340A clock generator can output four differential frequencies from 100Hz ~ 644.53125Mhz though I2C interface configuration. This section will show you how to use FPGA RTL IP to configure each Si5340A PLL and generate users desired output frequency to each peripheral.

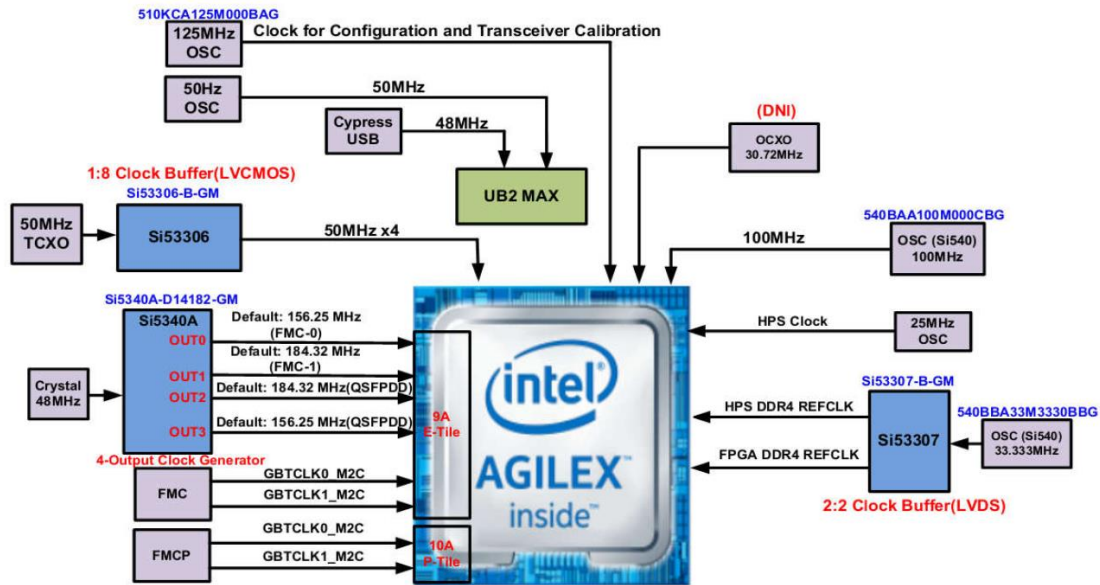


Figure 2-1 Clock tree of the Apollo Agilex SoM

■ Creating Si5340A Control IP

The Si5340A control IP is located in the folder: "\Demonstrations\FPGA\Si5340IP" in the System CD. Developers can use the IP directly in their Quartus top. Developers can refer to the example in Demonstrations/FPGA/Clock_Controller folder. This example shows how to instantiate the IP in Quartus top project.

■ Using Si5340 control IP

Table 2-1 lists the instruction ports of Si5340A Controller IP.

Table 2-1 Si5340A Controller Instruction Ports

Port	Direction	Description
iCLK	input	System Clock (50Mhz)
iRST_n	input	Synchronous Reset (0: Module Reset, 1: Normal)
iStart	input	Start to Configure (positive edge trigger)
iXCVR_REFCLK_0	input	Setting OUT0(FMC_REFCLK0_p) Channel Frequency Value

iXCVR_REFCLK_1	input	Setting OUT1(FMC_REFCLK1_p) Channel Frequency Value
iXCVR_REFCLK_2	input	Setting OUT2(QSFP28RSV_REFCLK_p) Channel Frequency Value
iXCVR_REFCLK_3	input	Setting OUT3(QSFP28_REFCLK_p) Channel Frequency Value
oPLL_REG_CONFIG_DONE	output	Si5340A Configuration status (0: Configuration in Progress, 1: Configuration Complete)
I2C_DATA	inout	I2C Serial Data to/from Si5340A
I2C_CLK	output	I2C Serial Clock to Si5340A

As shown in **Table 2-2**, the Si5340A control IP have preset several output frequency parameters, if users want to change frequency, users can fill in the input ports " iXCVR_REFCLK_0~3", with desired frequency values and recompile the project. For example, in the components Si5340A, change
. iXCVR_REFCLK_0 ('XCVR_REF_644M53125),
to. iXCVR_REFCLK_0 ('XCVR_REF_322M265625),

Recompile project, the Si5340A OUT0 channel (for FMC) output frequency will change from 644.53125Mhz to 322.26562Mhz.

Table 2-2 Si5340A Controller Reference Clock Frequency Setting for FMC/FMC+

iXCVR_REFCLK_0~3 Input Setting	Si5340A Channel Clock Frequency(MHz)
4'h0	644.53125
4'h1	625
4'h2	375
4'h3	322.265625
4'h4	312.5
4'h5	307.2
4'h6	250
4'h7	184.32
4'h8	156.25
4'h9	153.6

4'ha	125
4'hb	100

Users can also dynamically modify the input parameters, and input a positive edge trigger for “iStart”, then, Si5340A output frequency can be modified.

After the manually modifying, please remember to modify the corresponding frequency value in SDC file.

■ Modify Clock Parameter for Your Own Frequency

If the Si5340A control IP built-in frequencies are not users' desired, users can refer to the below steps to the modify control IP register parameter settings to modify the IP to output a desired frequency.

1. Firstly, download ClockBuilder Pro Software (See **Figure 2-2**), which is provided by Silicon Labs. This tool can help users to set the Si5340A's output frequency of each channel through the GUI interface, and it will automatically calculate the Register parameters required for each frequency. The tool download link: [http://url.terasic.com/clockuilder ro ofware](http://url.terasic.com/clockuilder_ro_ofware)

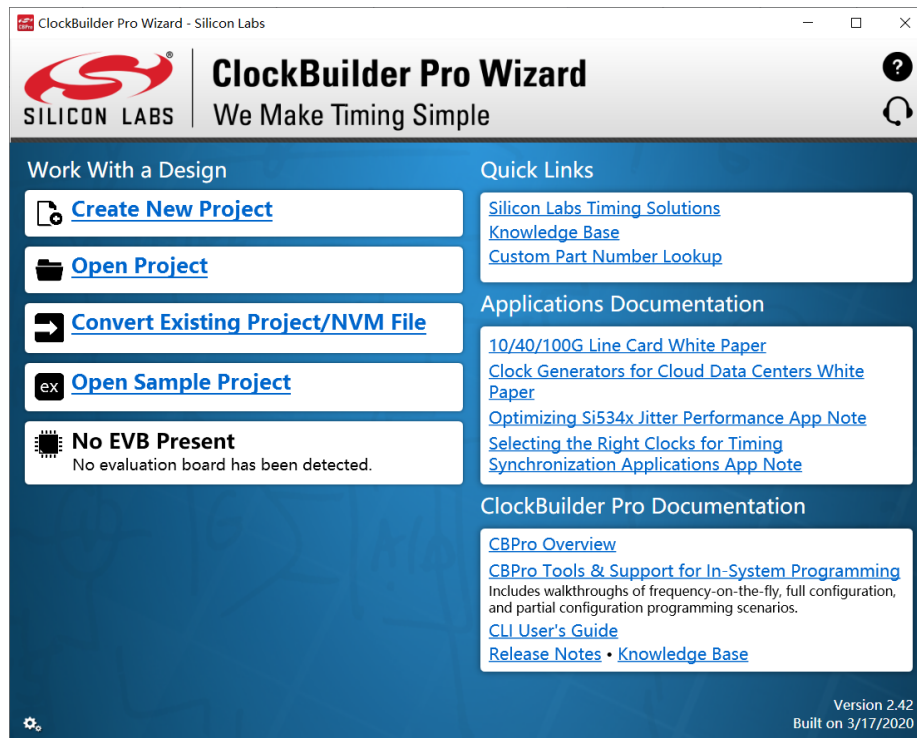


Figure 2-2 ClockBuilder Pro Wizard

2. After the installation, select Si5340, and configure the input frequency and output frequency as shown in **Figure 2-3**.

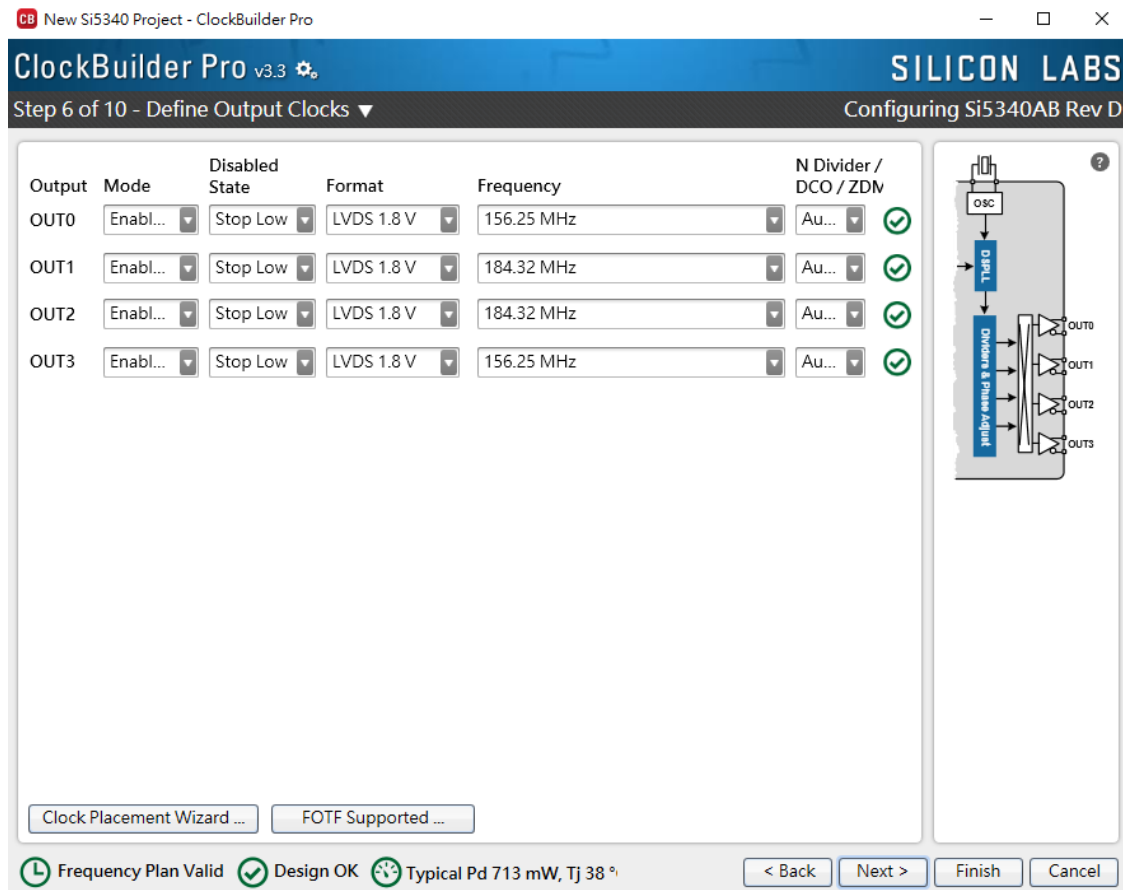


Figure 2-3 Define Output Clock Frequencies on ClockBuilder Pro Wizard

- After the setting is completed, ClockBuilder Pro Wizard generates a Design Report(text), which contains users setting frequency corresponding register value (See **Figure 2-4**).

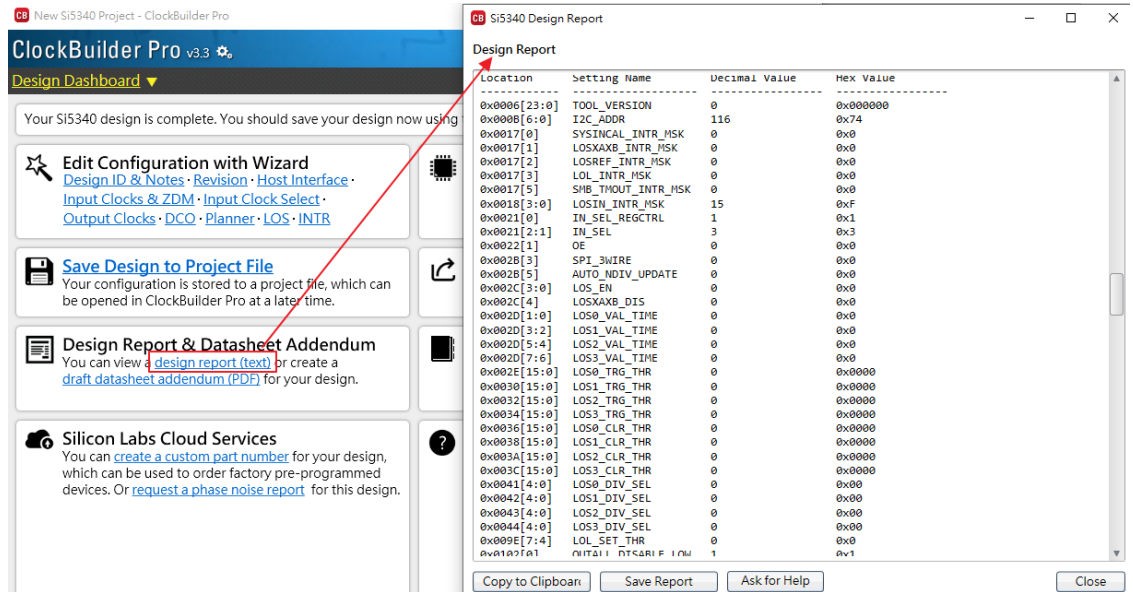


Figure 2-4 Open Design Report on ClockBuilder Pro Wizard

- Open Si5340 control IP sub-module “si5340a_i2c_reg_controller.v” as shown in **Figure 2-5**, refer to Design Report parameter to modify sub-module corresponding register value (See **Figure 2-6**).

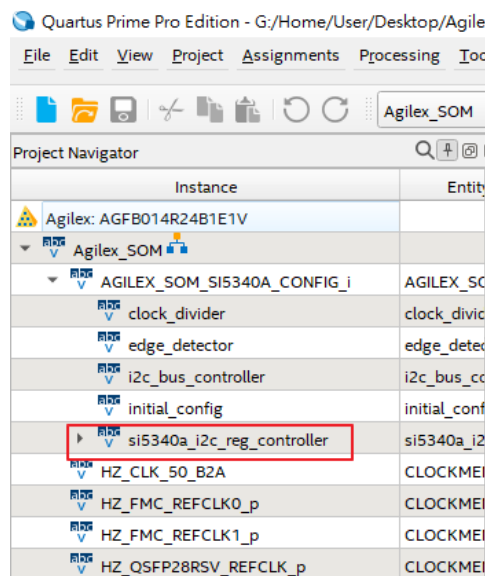


Figure 2-5 Sub-Module file "si5340a_i2c_reg_controller.v"

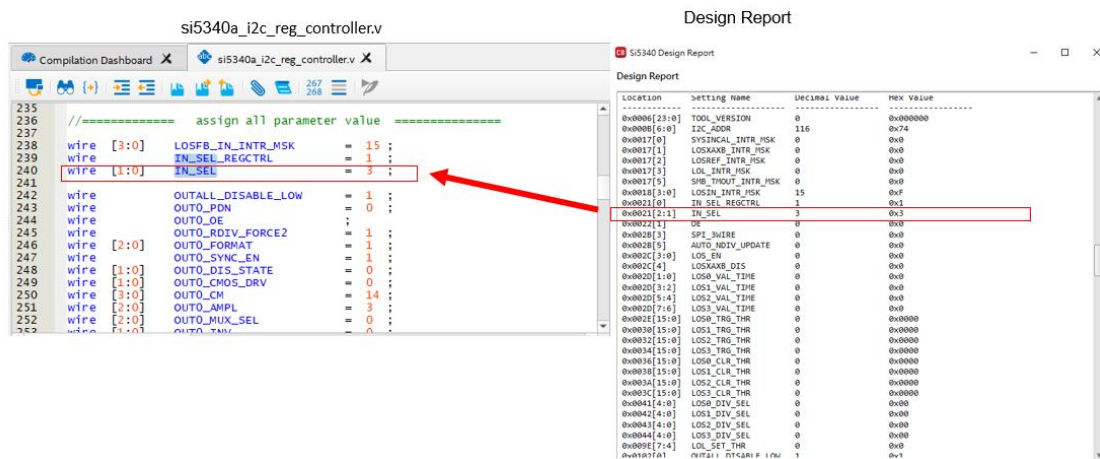


Figure 2-6 Modify Si5340A Control IP Base on Design Report

After modifying and compiling, Si5340A can output new frequencies according to the users' setting.

Note : No need to modify all Design Report parameters in `si5340a_i2c_reg_controller.v`, users can ignore parameters which have nothing to do with the frequency setting

2.2 Basic Nios II control demo for SI5340A/ Temperature/ Power/ Fan

This demonstration shows how to use the Nios II processor to program programmable clock generators (Si5340A) 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 monitoring fan rotation speed.

■ System Block Diagram

Figure 2-7 shows the system block diagram of this demonstration. The Si5340A clock generator is controlled through I2C controllers driven by Nios II program. The 12V input power monitor, temperature sensor and fan controller connected to the MAX10 FPGA and controlled by internal logic circuits. All collected status data or control commands will be sent to the SPI slave block so that the Agilix FPGA can read it through the SPI

interface.

In the Agilex FPGA, an SPI master IP (implemented by HDL) will read these external sensor data from the MAX10 FPGA through SPI interface. The Nios system will read these information or output the PLL control settings through PIO controllers.

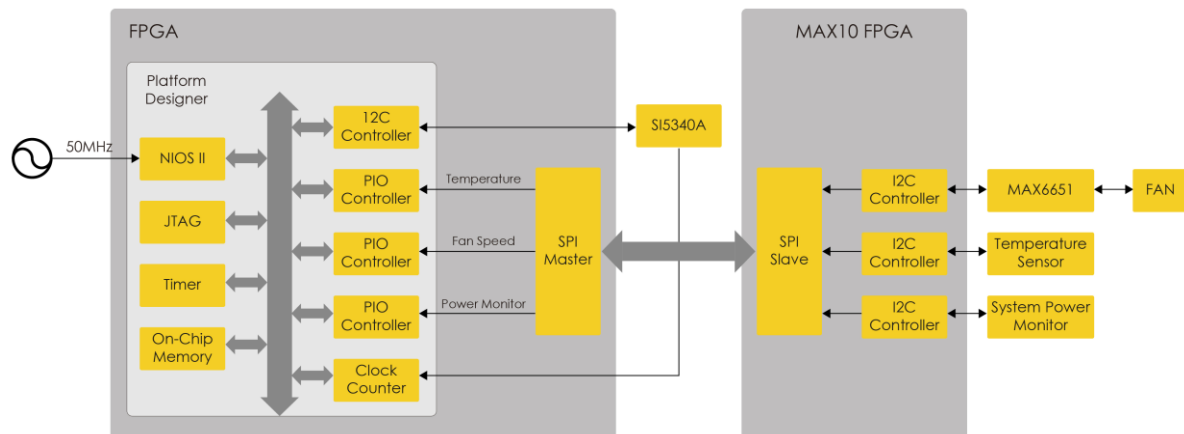


Figure 2-7 Block Diagram of the Nios II Basic Demonstration

The system provides a menu in nios-terminal, as shown in **Figure 2-8** to provide an interactive interface. With the menu, users can perform the test for the external programmable PLL and board info sensor. Note, pressing 'ENTER' should be followed with the choice number.

```
C:\intelFPGA_pro\21.4\quartus\bin64\nios2-terminal.exe
Info: System process ID: 1604
Info: Command: quartus_pgm -m jtag -c 1 -o p;Agilex_SOM.sof
Info (213045): Using programming cable "Apollo Agilex [USB-1]"
Info (213011): Using programming file Agilex_SOM.sof with checksum 0x3F4236AD for device AGFB014R24B@1
Info (209060): Started Programmer operation at Wed Jul 13 02:18:53 2022
Info (18942): Configuring device index 1
Info (18943): Configuration succeeded at device index 1
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Wed Jul 13 02:18:57 2022
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1788 megabytes
Info: Processing ended: Wed Jul 13 02:18:57 2022
Info: Elapsed time: 00:00:15
Info: System process ID: 1604
Using cable "Apollo Agilex [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 126KB in 0.1s
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x00080238
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Apollo Agilex [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

===== Agilex Demo Program =====
[0] Si5340A
[1] Display Board Info
Input your choice: _
```

Figure 2-8 Menu of Demo Program

In board info test, the program will display local temperature, remote temperature, 12V input power monitor and fan rotation speed. The remote temperature is the FPGA temperature, and the local temperature is the board temperature where the temperature sensor located. A power monitor IC (LTC2945) embedded on the board can monitor real-time current and power. This IC can work out current/power value as multiplier and divider are embedded in it. There is a sense resistor R5 (0.003 Ω) for LTC2945 in the circuit, when power on the Apollo Agilex, there will be a voltage drop (named Δ SENSE Voltage) on R5. Based on sense resistors, the program of power monitor can calculate the associated voltage, current and power consumption.

In the external PLL programming test, the program will program the PLL first, and subsequently use TERAASIC custom Platform Designer CLOCK_COUNTER IP to count the clock count in a specified period to check whether the output frequency as changed as configured. For Si5340A programming, please note the device I2C address is 0xEE. The program can control the Si5340A to configure the output frequency of FMC_REFCLK0, FMC_REFCLK1, QSFP28 and QSFP28RSV according to your choice.

■ Demonstration File Location

- Hardware project directory: NIOS_BASIC_DEMO

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

■ Demonstration Setup and Instructions

1. Make sure Quartus Prime is installed on the Host PC.
2. Power on the FPGA board.
3. Use the USB Cable to connect your PC and the FPGA board and install USB Blaster II driver if necessary.
4. Execute the demo batch file “test.bat” under the batch file folder: NIOS_BASIC_DEMO\demo_batch.
5. After the Nios II program is downloaded and executed successfully, a prompt message will be displayed in nios2-terminal.
6. For the PLL Si5340A test, please input key ‘0’ and input the desired output frequency for eight clock sources, as shown in **Figure 2-9**.

```

C:\intelFPGA_pro\21.4\quartus\bin64\nios2-terminal.exe
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Wed Jul 13 03:00:19 2022
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
===== Si5340A Programming =====
[0] 644.531250 MHz
[1] 625.000000 MHz
[2] 375.000000 MHz
[3] 322.265625 MHz
[4] 312.500000 MHz
[5] 307.200012 MHz
[6] 250.000000 MHz
[7] 184.320007 MHz
[8] 156.250000 MHz
[9] 153.600006 MHz
[10] 125.000000 MHz
[11] 100.000000 MHz
[12] 0.000000 MHz
[Other] exit
please select FMC REF CLK0:1
please select FMC REF CLK1:2
please select QSF28RSV:3
please select QSF28:4

I2C core is enabled!
successFMC REF CLK0_REFCLK PASS (clk1=998, clk2=12475)
FMC REF CLK1_REFCLK PASS (clk1=998, clk2=7485)
QSF28RSV_REFCLK PASS (clk1=998, clk2=6433)
QSF28_REFCLK PASS (clk1=998, clk2=6237)
Si5340A Test:PASS
===== Agilex Demo Program =====
[0] Si5340A
[1] Display Board Info
Input your choice:

```

Figure 2-9 Si5340A Demo

7. For temperature, power monitor and fan test, please input key ‘1’ and press ‘Enter’ in the nios-terminal, as shown in **Figure 2-10**.

```
C:\intelFPGA_pro\21.4\quartus\bin64\nios2-terminal.exe
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

===== Agilex Demo Program =====
[0] Si5340A
[1] Display Board Info
Input your chioce:1
==== Temperature ====
    FPGA: 45°C
    Board 1: 38°C
    Board 2: 39°C
    SDM: 45°C
    E-Tile: 47°C
    P-Tile: 46°C

==== Fan ====
    Fan 1 RPM: 2790

==== Power (12V) Monitor ====
    Voltage      = 12.050 V
    Current       = 1.616 A
    Power         = 19.473 W
==== Core Power Monitor ====
    Voltage      = 0.868 V
    Current       = 2.550 A
    Power         = 2.213 W
Display Board Info Test:PASS
===== Agilex Demo Program =====
[0] Si5340A
[1] Display Board Info
Input your chioce:_
```

Figure 2-10 Board Info Demo

2.3 DDR4 SDRAM RTL Test

This demonstration performs a memory test function using RTL code on the two DDR4 SO-DIMM on the Apollo Agilex board. Both DDR4 SO-DIMM are controlled by FPGA fabric. The memory size of each DDR4 SO-DIMM SDRAM used in this test is 16 GB.

■ Function Block Diagram

Figure 2-11 shows the function block diagram of this demonstration. There are two DDR4 SDRAM controllers (DDR4A and DDR4B) in this project. All of the controllers use 33.333 MHz as a reference clock. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-3**. The test program will write data into SDRAM, after writing 16GB capacity, it will read the values from SDRAM and check whether there is any error.

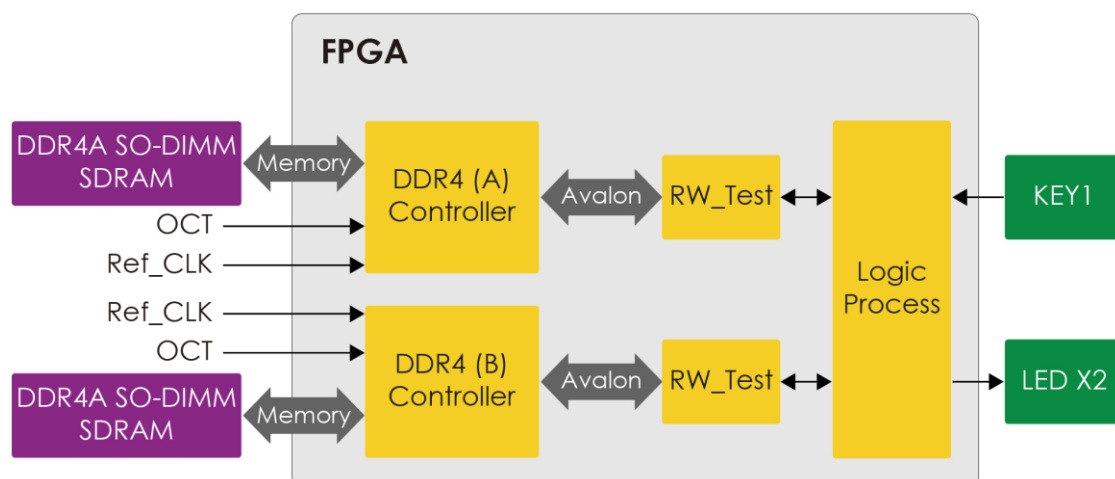


Figure 2-11 Block diagram of the DDR4 RTL demonstration

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

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
AGFB014R24B1E1V	1600 (DDR4 3200)
AGFB014R24B2E2V	1333 (DDR4 2666)

■ Agilex External Memory Interfaces

To use Agilex External Memory Interfaces controller for DDR4 SO-DIMM SDRAM, please perform the two major steps below:

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

■ Design Tools

- Quartus Prime 21.4 Pro Edition or later

■ Demonstration Source Code

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

The demo batch file includes following files:

- ◆ Batch File: test.bat

■ Demonstration Setup

1. Make sure Quartus Prime Pro Edition is installed on the Host PC.
2. Connect the Apollo Agilex board to the Host PC via the USB cable. Install the USB-Blaster II driver if necessary.
3. Power on the Apollo Agilex board.
4. Execute the demo batch file “test.bat” under the batch file folder \RTL_DDR4_Test\demo_batch.
5. Press **KEY1** (see [Figure 2-12](#)) to start DDR4 write & loopback verify process. It will take about 1 second to perform the test. While testing, the LED will blink. When LED stop blinking it means the test process is done.
6. The test result will show on LED0 and LED1. The **LED0** represents the test result for the DDR4A, the **LED1** represents the test result for the DDR4B. If the LED0/LED1 light on, it means the test result is passed. If the LED0/LED1 is off, it means the test result is failed.
7. User can press **KEY1** again to regenerate the test control signals for a repeat test.

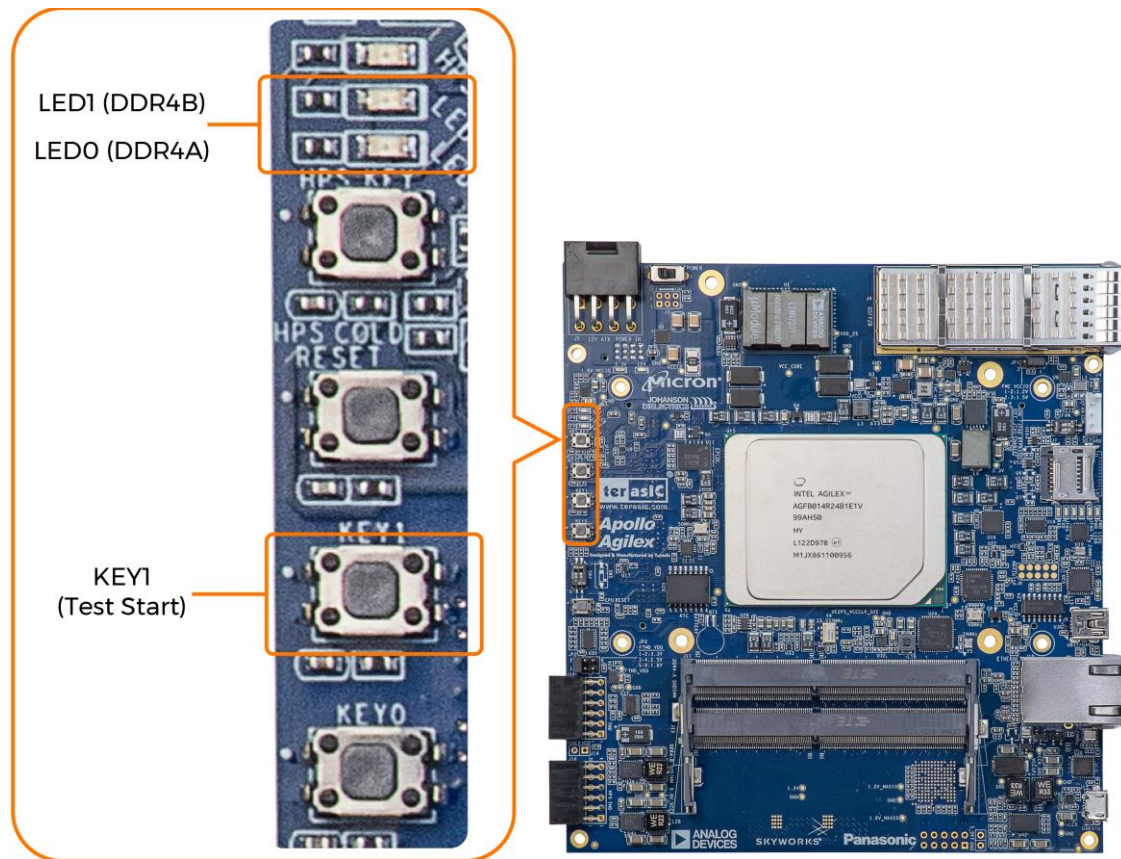


Figure 2-12 Location of the KEY and LED on the Apollo Agilex board

2.4 DDR4 SDRAM Test by Nios II

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

■ System Block Diagram

Figure 2-13 shows the system block diagram of this demonstration. In the Platform

Designer (formerly Qsys), one 50 MHz, dual frequency OSC and PLL clock generator(Si5340A) are used. The Si5340A and dual frequency OSC will provide 33.333Mhz clock to the DDR4A and DDR4B SO-DIMM socket as the reference clock. There are two DDR4 Controllers which are used in the demonstrations. Each controller is responsible for one DDR4 SO-DIMM socket (DDR4A and DDR4B). Each DDR4 controllers are configured as 16GB DDR4 controller. Depending on the FPGA with different speed grade, the controller generates clocks with different speeds. For details, please refer to **Table 2-4**. The Nios II processor is used to perform the memory test. The Nios II program is running in the On-Chip Memory. A PIO Controller is used to monitor buttons status which is used to trigger starting memory testing.

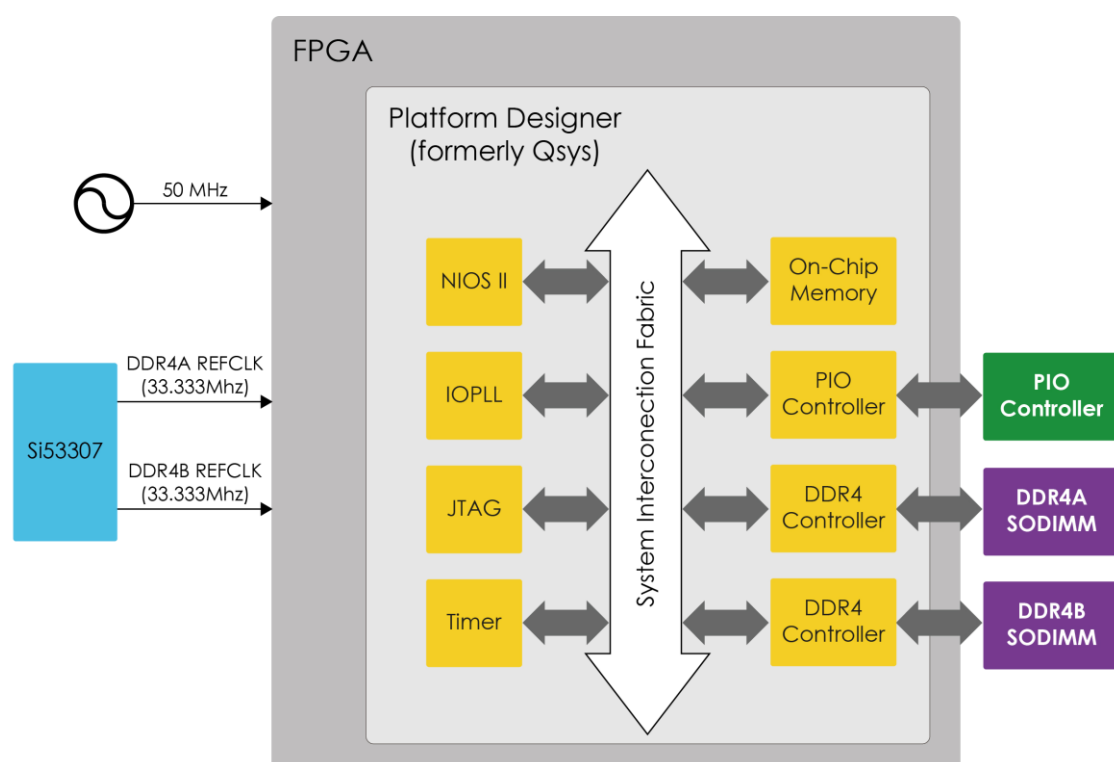


Figure 2-13 Block diagram of the DDR4 Basic Demonstration

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

completed, the result is displayed in the Nios II terminal.

Table 2-4 DDR4 clock frequency for each speed grade of FPGA

FPGA Speed Grade	DDR4 Clock Frequency(MHz)
AGFB014R24B1E1V	1600 (DDR4 3200)
AGFB014R24B2E2V	1333 (DDR4 2666)

■ Design Tools

- Quartus Prime 21.4 Pro Edition

■ Demonstration Source Code

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

■ Nios II Project Compilation

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

■ Demonstration Batch File

Demo Batch File Folder: NIOS_DDR4_Test\demo_batch

The demo batch file includes following files:

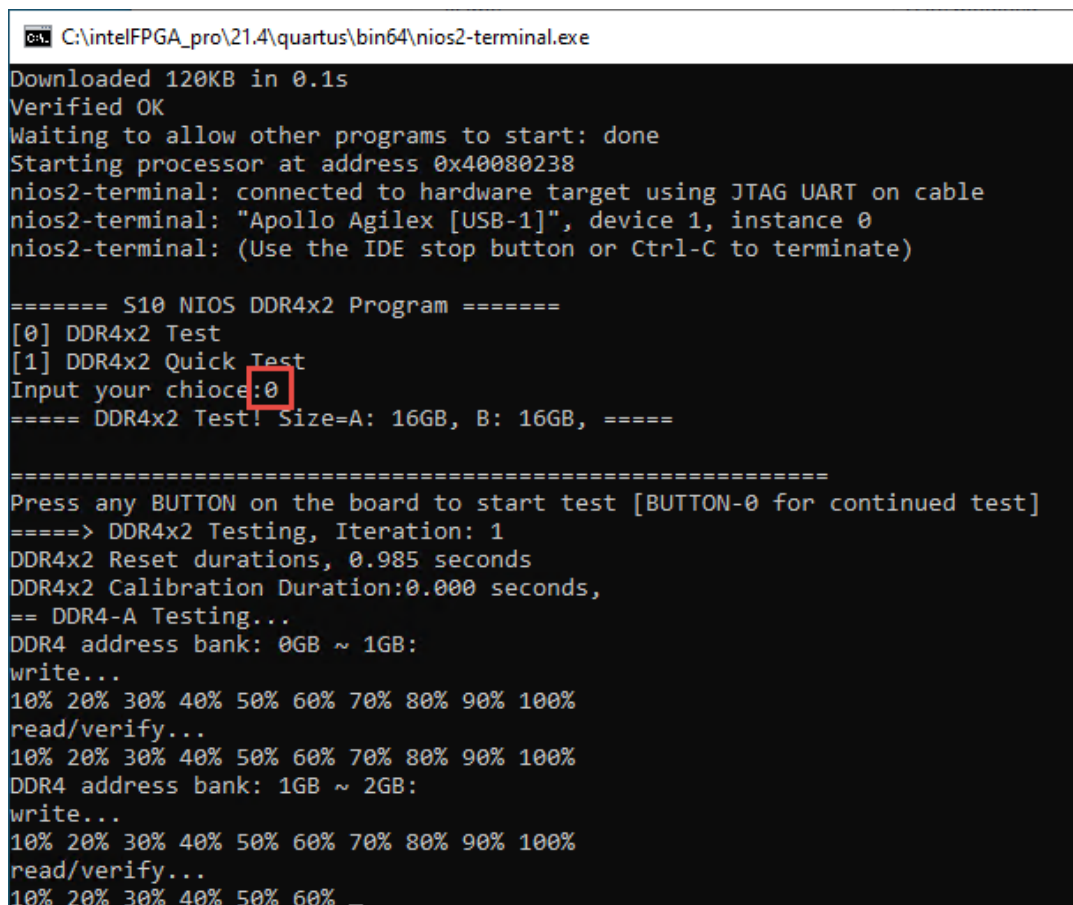
- Batch File for USB-Blaster II: test.bat, test.sh
- FPGA Configure File: Agilex_SOM.sof
- Nios II Program: MEM_TEST.elf

■ Demonstration Setup

Please follow below procedures to set up the demonstrations.

1. Make sure Quartus Prime and Nios II are installed on your PC.
2. Power on the FPGA board.
3. Use a USB Cable to connect the PC and the FPGA board and install USB Blaster II driver if necessary.

4. Execute the demo batch file "test.bat" under the folder "NIOS_DDR4_Test\demo_batch".
5. After the Nios II program is downloaded and executed successfully, a prompt message will be displayed in the nios2-terminal.
6. For DDR4 test, please input key '0' and press 'Enter' in the nios2-terminal as shown in **Figure 2-14**. The program will display progressing and result information.
7. For DDR4 quick test, please input key '1' and press 'Enter' in the nios2-terminal as shown in **Figure 2-15**. The program will display progressing and result information. Press Button0~Button1 of the FPGA board to start SDRAM verify process, and press Button0 for continued test.



```

C:\intelFPGA_pro\21.4\quartus\bin64\nios2-terminal.exe
Downloaded 120KB in 0.1s
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x40080238
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Apollo Agilex [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

===== S10 NIOS DDR4x2 Program =====
[0] DDR4x2 Test
[1] DDR4x2 Quick Test
Input your chioce:0
===== DDR4x2 Test! Size=A: 16GB, B: 16GB, =====

=====
Press any BUTTON on the board to start test [BUTTON-0 for continued test]
=====> DDR4x2 Testing, Iteration: 1
DDR4x2 Reset durations, 0.985 seconds
DDR4x2 Calibration Duration:0.000 seconds,
== DDR4-A Testing...
DDR4 address bank: 0GB ~ 1GB:
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
DDR4 address bank: 1GB ~ 2GB:
write...
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
read/verify...
10% 20% 30% 40% 50% 60%

```

Figure 2-14 Progress option [0] DDR4x2 Test

```

C:\intelFPGA_pro\21.4\quartus\bin64\nios2-terminal.exe
Using cable "Apollo Agilex [USB-1]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 120KB in 0.1s
Verified OK
Waiting to allow other programs to start: done
Starting processor at address 0x40080238
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "Apollo Agilex [USB-1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

===== S10 NIOS DDR4x2 Program =====
[0] DDR4x2 Test
[1] DDR4x2 Quick Test
Input your choice: 1
===== DDR4x2 Test! Size=A: 16GB, B: 16GB =====

=====
Press any BUTTON on the board to start test [BUTTON-0 for continued test]
=====> DDR4x2 Testing, Iteration: 1
DDR4x2 Reset durations, 0.998 seconds
DDR4x2 Calibration Duration:0.000 seconds,
== DDR4-A Testing...
DDR4A address bank: 0GB ~ 1GB: PASS
DDR4A address bank: 1GB ~ 2GB: PASS
DDR4A address bank: 2GB ~ 3GB: PASS
DDR4A address bank: 3GB ~ 4GB: PASS
DDR4A address bank: 4GB ~ 5GB: PASS
DDR4A address bank: 5GB ~ 6GB: PASS
DDR4A address bank: 6GB ~ 7GB: PASS
DDR4A address bank: 7GB ~ 8GB: PASS
DDR4A address bank: 8GB ~ 9GB: PASS
DDR4A address bank: 9GB ~ 10GB: PASS
DDR4A address bank: 10GB ~ 11GB: PASS
DDR4A address bank: 11GB ~ 12GB: PASS
DDR4A address bank: 12GB ~ 13GB: PASS
DDR4A address bank: 13GB ~ 14GB: PASS
DDR4A address bank: 14GB ~ 15GB: PASS
DDR4A address bank: 15GB ~ 16GB: PASS
DDR4A test:Pass, 51 seconds
== DDR4-B Testing...
DDR4B address bank: 0GB ~ 1GB: PASS
DDR4B address bank: 1GB ~ 2GB: PASS
DDR4B address bank: 2GB ~ 3GB: PASS
DDR4B address bank: 3GB ~ 4GB: PASS
DDR4B address bank: 4GB ~ 5GB: PASS
DDR4B address bank: 5GB ~ 6GB: PASS
DDR4B address bank: 6GB ~ 7GB: PASS
DDR4B address bank: 7GB ~ 8GB: PASS
DDR4B address bank: 8GB ~ 9GB: PASS
DDR4B address bank: 9GB ~ 10GB: PASS
DDR4B address bank: 10GB ~ 11GB: PASS
DDR4B address bank: 11GB ~ 12GB: PASS
DDR4B address bank: 12GB ~ 13GB: PASS
DDR4B address bank: 13GB ~ 14GB: PASS

```

Figure 2-15 Progress and Result Information for “DDR4 Quick Test”

2.5 Board Information IP

This section will introduce an IP which can be placed in the Agilex FPGA and allows users to obtain board status information such as power, temperature, and fan speed on the Apollo Agilex board.

The Apollo Agilex board provides several sensors to monitor the status of the board, such as Agilex FPGA temperature, board power monitor, and fan speed status. These interfaces are connected to the system MAX FPGA on the board. The logic in the system MAX FPGA will automatically read the status values of these sensors and store them in the internal register. As shown in **Figure 2-16**, there is an SPI slave IP in the system MAX FPGA will read the value of the board status from these registers and it can be output to SPI master logic via SPI interface.

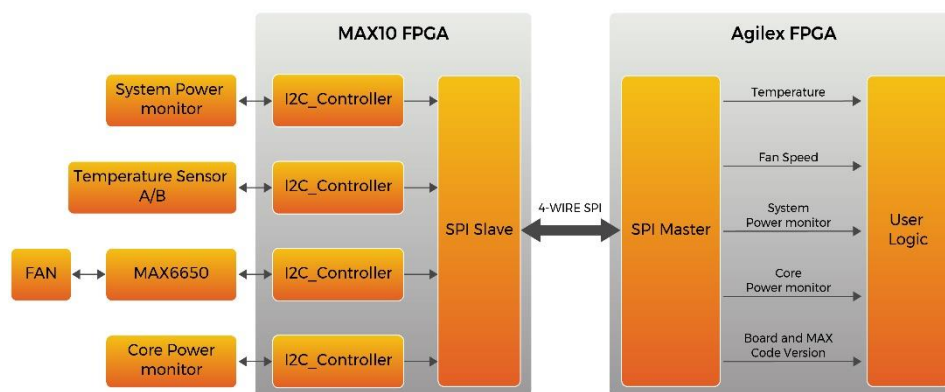


Figure 2-16 the board information IP architecture

User can placing a board information IP (BOARD_INFO.v ; SPI master) provided by Terasic in the Agilex FPGA, the board status can be obtained via SPI interface from the system MAX FPGA and output to user logic.

The board information IP can be obtained from the following path in the system CD:
System CD/Demonstration/FPGA/NIOS_BASIC_DEMO/SPI/BOARD_INFO.v

Figure 2-17 shows the input and output pins of the board information IP. Detailed pin descriptions and functions can be obtained from **Table 2-5** Board information IP input and output ports. The user only needs to provide the IP 50Mhz clock and the reset

control signal. The IP will automatically communicate with the system MAX FPGA to get the board status value via the SPI interface. When the logic level of the **Info_Valid** signal is from low to high, it means that the board status has been updated and can be used.

Finally, **Figure 2-18** shows the status of the IP during execution.

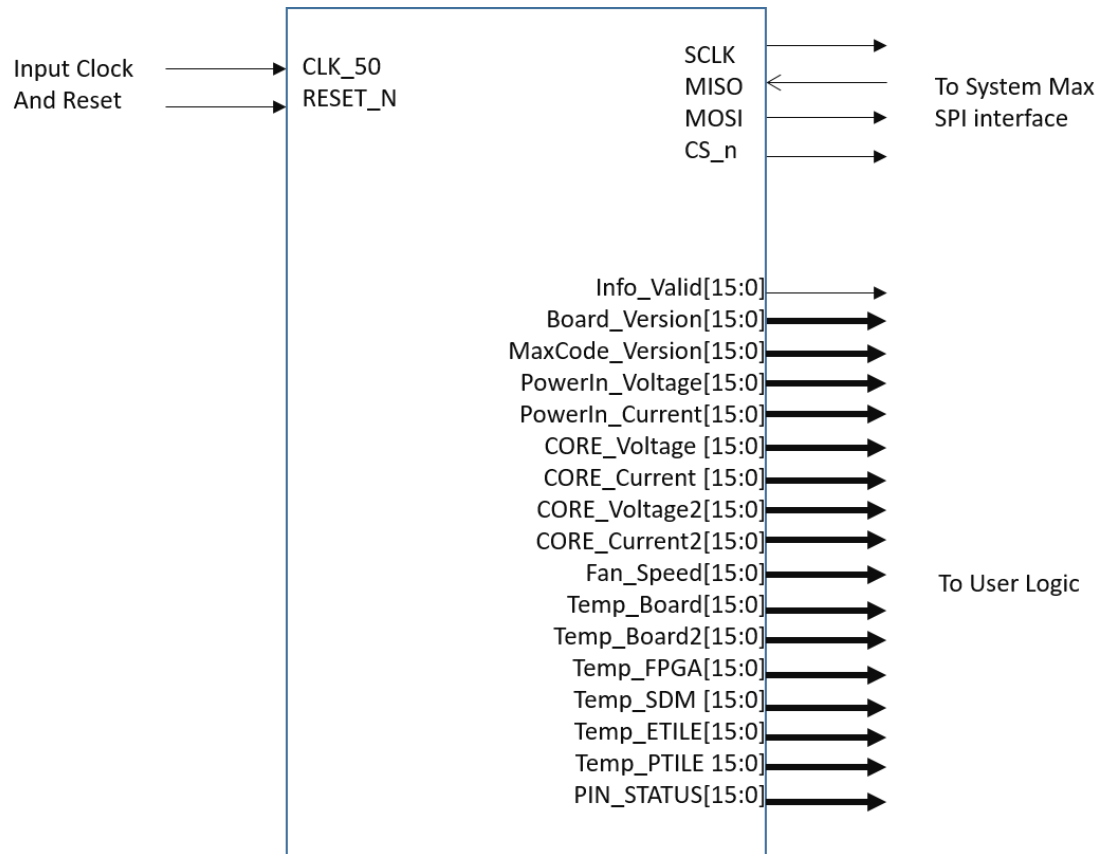


Figure 2-17 Pin out of the board information IP

Table 2-5 Board information IP input and output ports

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

SCLK	Output	1	Serial Clock, SPI master output to slave. Please connect this signal to the INFO_SPI_SCLK pin.
Info_Valid	Output	1	Information valid, logic high indicates board status updated ready.
Board_Version	Output	16	This information indicates the version of the DE10-Agilex board. It will be started at 0x000A.
MaxCode_Version	Output	16	This information indicates the version of the System MAX 10 FPGA code. It will be started at 0x0001.
PowerIn_Voltage	Output	16	12V Voltage, the unit of the output value is mV. If the PowerIn_Voltage output value is "12050" that means 12.05V for 12V power
PowerIn_Current	Output	16	Current of the 12V power, the unit of the output value is mA. If the PowerIn_Current
CORE_Voltage	Output	16	Core voltage of the first power channel , Unit is mV
CORE_Current	Output	16	Current of the first power channel , Unit is mA
CORE_Voltage2	Output	16	Core voltage of the second power channel , Unit is mV
CORE_Current2	Output	16	Current of the second power channel , Unit is mA
Fan_Speed	Output	16	First fan speed of the board. The unit of the output value is RPM.
Temp_Board	Output	16	First ambient temperature of the development board. The unit of the output value is Celsius.
Temp_Board2	Output	16	Second ambient temperature of the development board. The unit of the output value is Celsius.
Temp_FPGA	Output	16	Core FPGA temperature of the development board. The unit of the output value is Celsius.
Temp_SDM	Output	16	SDM FPGA temperature of the development board. The unit of the output value is Celsius

Temp_ETILE	Output	16	Temperature of the E-TILE transceiver in the FPGA. The unit of the output value is Celsius
Temp_PTILE	Output	16	Temperature of the P-TILE transceiver in the FPGA. The unit of the output value is Celsius
PIN_STATUS	Output	16	<p>BIT8~15 : Reserved to 0.</p> <p>BIT7 : FAN_ALERT_n, When the fan speed is abnormal, this bit is 0.</p> <p>BIT6 : Reserved to 1.</p> <p>BIT5: When shutdown occurs, this bit is 0.</p> <p>BIT4: Reserved to 1.</p> <p>BIT3: Reserved to 0.</p> <p>BIT2: FPGA_CONF_DONE,FPGA Configure success, this bit is 1.</p> <p>BIT1: Reserved to 1.</p> <p>BIT0: Reserved to 1..</p>

Pre-Syn	BOARD_INFO_0[Board_Version[15..0]	000Ah
Pre-Syn	BOARD_INFO_1[MaxCode_Version[15..0]	0001h
Pre-Syn	BOARD_INFO_2[PowerIn_Voltage[15..0]	11775
Pre-Syn	BOARD_INFO_3[PowerIn_Current[15..0]	1616
Pre-Syn	BOARD_INFO_4[CORE_Voltage[15..0]	868
Pre-Syn	BOARD_INFO_5[CORE_Current[15..0]	1378
Pre-Syn	BOARD_INFO_6[Fan_Speed[15..0]	2520
Pre-Syn	BOARD_INFO_7[Temp_FPGA[15..0]	38
Pre-Syn	BOARD_INFO_8[Temp_Board[15..0]	29
Pre-Syn	BOARD_INFO_9[PIN_STATUS[15..0]	00F7h
Pre-Syn	BOARD_INFO_10[Temp_SDM[15..0]	39
Pre-Syn	BOARD_INFO_11[Temp_ETILE[15..0]	40
Pre-Syn	BOARD_INFO_12[Temp_PTILE[15..0]	39
Pre-Syn	BOARD_INFO_13[CORE_Voltage2[15..0]	868
Pre-Syn	BOARD_INFO_14[CORE_Current2[15..0]	853
Pre-Syn	BOARD_INFO_15[Temp_Board2[15..0]	30
Pre-Syn	BOARD_INFO_16[Info_Valid	

Figure 2-18 Waveform of the board status output

2.6 100G Ethernet Example

This 100G Ethernet example is generated according to the documents [E-Tile Ethernet IPfor Intel Agilex FPGA Design Example](#). The E-Tile Ethernet IP is used in the example design. The IP is configured as 100GE MAC+PC with (528,514) RS-FEC. This example executes the internal and external loopback test through four-channel of one QSFP28 ports on the FPGA main board. For external loopback test, a QSFP28 loopback fixture is required, otherwise only internal loopback test be available. **Figure 2-19** shows the block diagram of this demonstration.

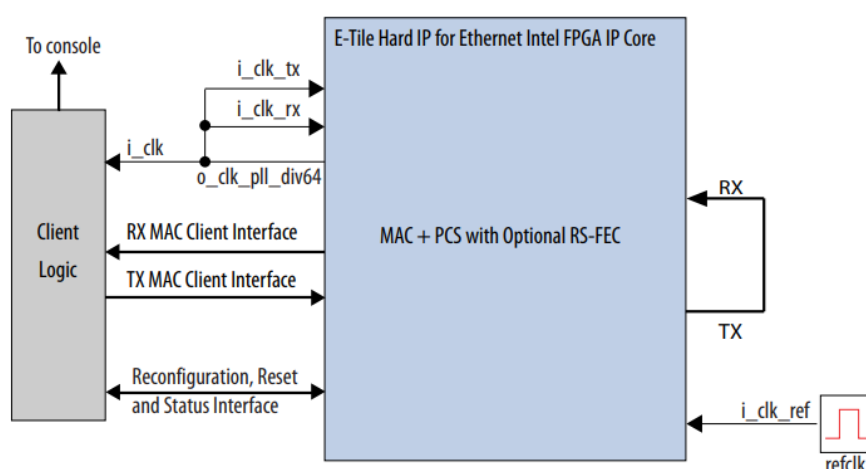


Figure 2-19 Block diagram of 100GbE demo

■ Project Information

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

Item	Description
Project Location	Ethernet_100G
Quartus Project	Ethernet_100G\hardware_test_design
FPGA Bit Stream	Ethernet_100G\demo_batch
Test Scrip File	Ethernet_100G\hardware_test_design\hwtest\main.tcl
Quartus Version	Quartus Prime 21.4 Pro Edition

Figure 2-20 shows the IP setup for the demonstration. **Single 100GE with optional**

RSFEC Core Variant is selected and **Enable RSFEC** is checked.

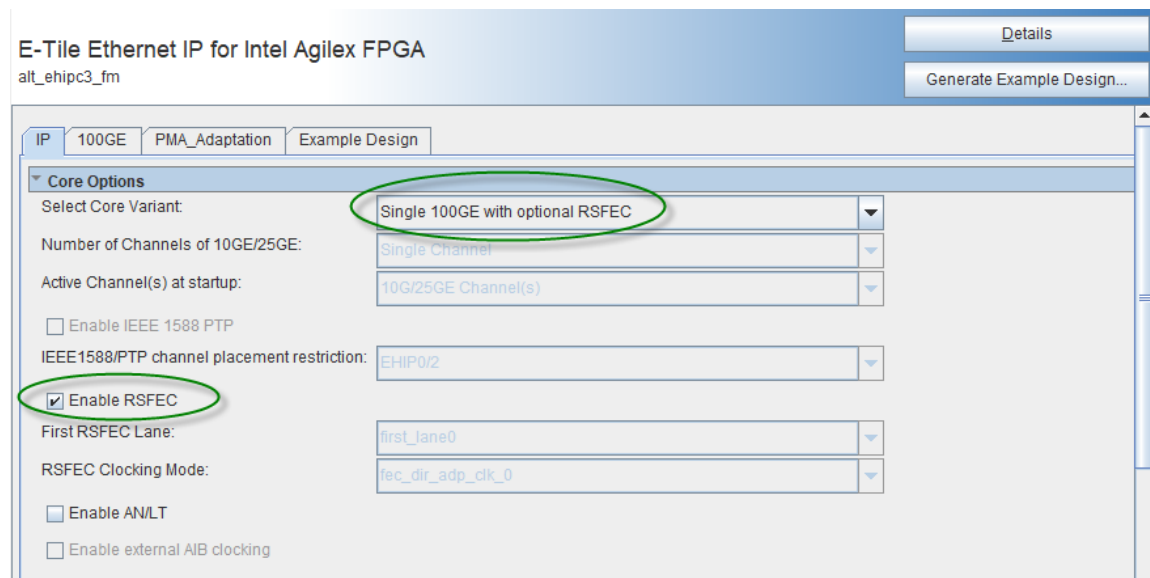


Figure 2-20 Core Variant Setup for Ethernet IP

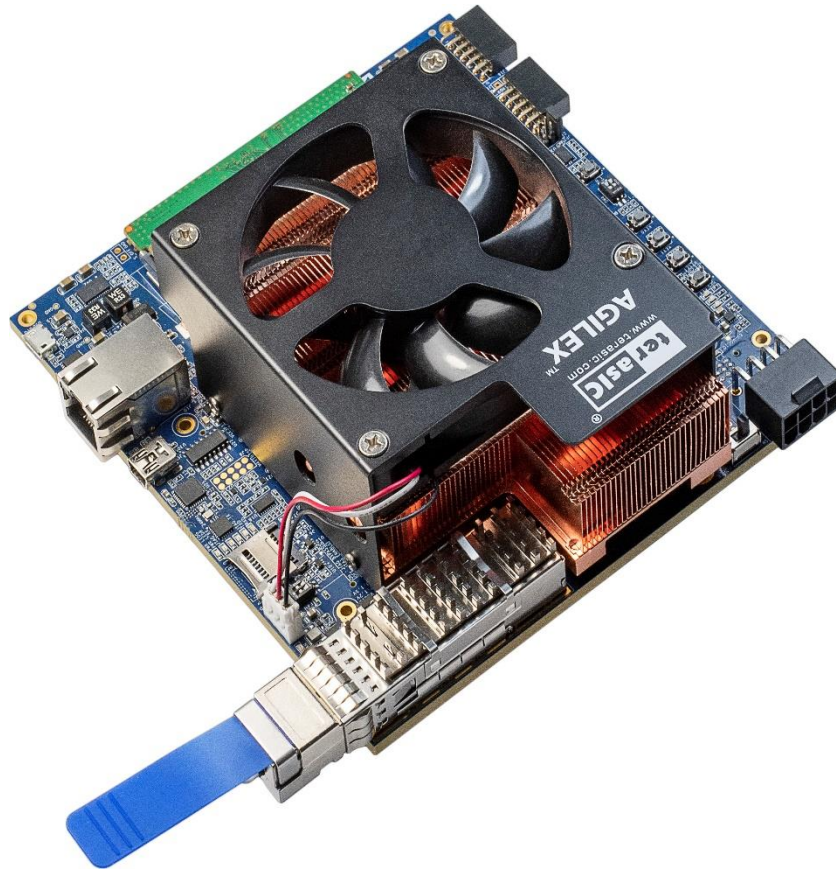
Four transceiver channels are used in the Quartus Project. The pre-compiled .sof files on the demo_batch folder.

■ Demonstration Setup

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

1. Insert a QSFP28 loopback fixture into the QSFP28 port on the Apollo-Som board, as shown in **Figure 2-21**.
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. Goto demo_batch folder and execute test.bat to configure FPGA. There are four pre-compiled sof files are available.
4. Open alt_ehipc3_fm_100GE Quartus Project and launch the System Console by selecting the menu item **Tools → System Debugging Tools → System Console** in Quartus.

5. In the System Console window, input the following commands to start the loopback test, as shown in **Figure 2-22**.
`%cd hwtest`
`%source main.tcl`
`%run_test_ex`
6. The loopback test report will be displayed in the Tcl Console, as shown in **Figure 2-23** and **Figure 2-24**.



**Figure 2-21 Setup QSFP28
loopback fixture**

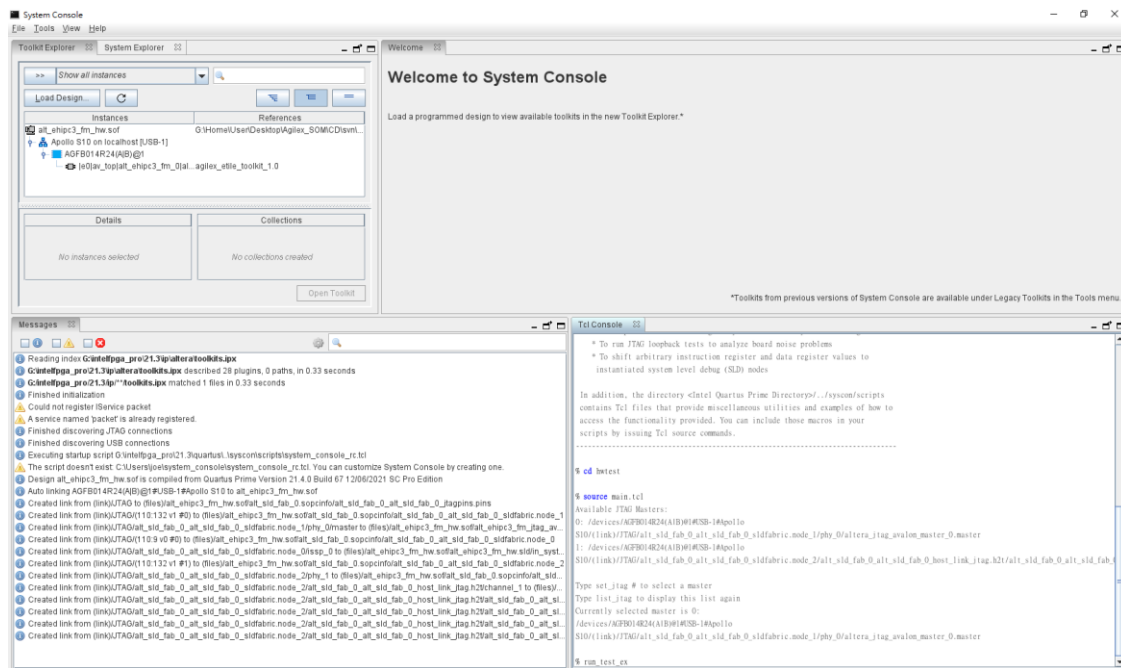


Figure 2-22 Launch the System Console for Ethernet 100G Demo

STATISTICS FOR BASE 2304 (Rx)	
Fragmented Frames	: 0
Jabbered Frames	: 0
Any Size with FCS Err Frame	: 1
Right Size with FCS Err Fra	: 1
Multicast data Err Frames	: 0
Broadcast data Err Frames	: 0
Unicast data Err Frames	: 1
Multicast control Err Frame	: 0
Broadcast control Err Frame	: 0
Unicast control Err Frames	: 0
Pause control Err Frames	: 0
64 Byte Frames	: 7298
65 - 127 Byte Frames	: 7024
128 - 255 Byte Frames	: 14578
256 - 511 Byte Frames	: 28646
512 - 1023 Byte Frames	: 57403
1024 - 1518 Byte Frames	: 55934
1519 - MAX Byte Frames	: 0
> MAX Byte Frames	: 1670922
Rx Frame Starts	: 1841805
Multicast data OK Frame	: 0
Broadcast data OK Frame	: 0
Unicast data OK Frames	: 1837811
Multicast Control Frames	: 0
Broadcast Control Frames	: 0
Unicast Control Frames	: 0
Pause Control Frames	: 0

Figure 2-23 Ethernet 100G loopback test report for RX

Tcl Console	
STATISTICS FOR BASE 2048 (Tx)	
Fragmented Frames	: 0
Jabbered Frames	: 0
Any Size with FCS Err Frame	: 1
Right Size with FCS Err Fra	: 1
Multicast data Err Frames	: 0
Broadcast data Err Frames	: 0
Unicast data Err Frames	: 1
Multicast control Err Frame	: 0
Broadcast control Err Frame	: 0
Unicast control Err Frames	: 0
Pause control Err Frames	: 0
64 Byte Frames	: 7298
65 - 127 Byte Frames	: 7024
128 - 255 Byte Frames	: 14578
256 - 511 Byte Frames	: 28646
512 - 1023 Byte Frames	: 57403
1024 - 1518 Byte Frames	: 55934
1519 - MAX Byte Frames	: 0
> MAX Byte Frames	: 1670922
Tx Frame Starts	: 1841805
Multicast data OK Frame	: 0
Broadcast data OK Frame	: 0
Unicast data OK Frames	: 1837811
Multicast Control Frames	: 0
Broadcast Control Frames	: 0
Unicast Control Frames	: 0
Pause Control Frames	: 0
0	
%	

Figure 2-24 Ethernet 100G loopback test report for TX

2.7 24G CPRI Example

This 24G CPRI example is generated according to the documents [E-Tile Ethernet IP for Intel Agilex FPGA Design Example](#). The E-Tile CPRI IP is used in the example design. The IP is configured as 24G. This example executes the internal and external loopback test through four-channel of one QSFP28 ports on the FPGA main board. For external loopback test, a QSFP28 loopback fixture is required, otherwise only internal loopback test be available. **Figure 2-25** shows the block diagram of this demonstration.

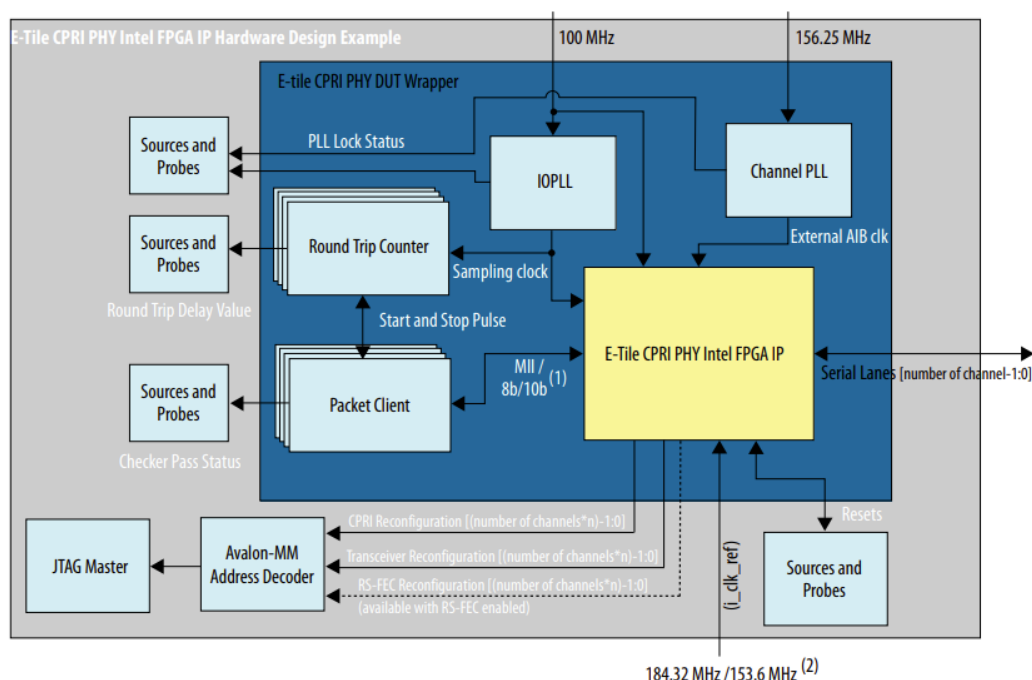


Figure 2-25 Block diagram of 24G CPRI demo

■ Project Information

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

Item	Description
Project Location	CPRI
Quartus Project	CPRI\hardware_test_design
FPGA Bit Stream	CPRI\demo_batch
Test Scrip File	CPRI\hardware_test_design\hwtest\main.tcl
Quartus Version	Quartus Prime 22.1 Pro Edition

Figure 2-26 shows the IP setup for the demonstration. **Single 24G** Core Variant is selected .

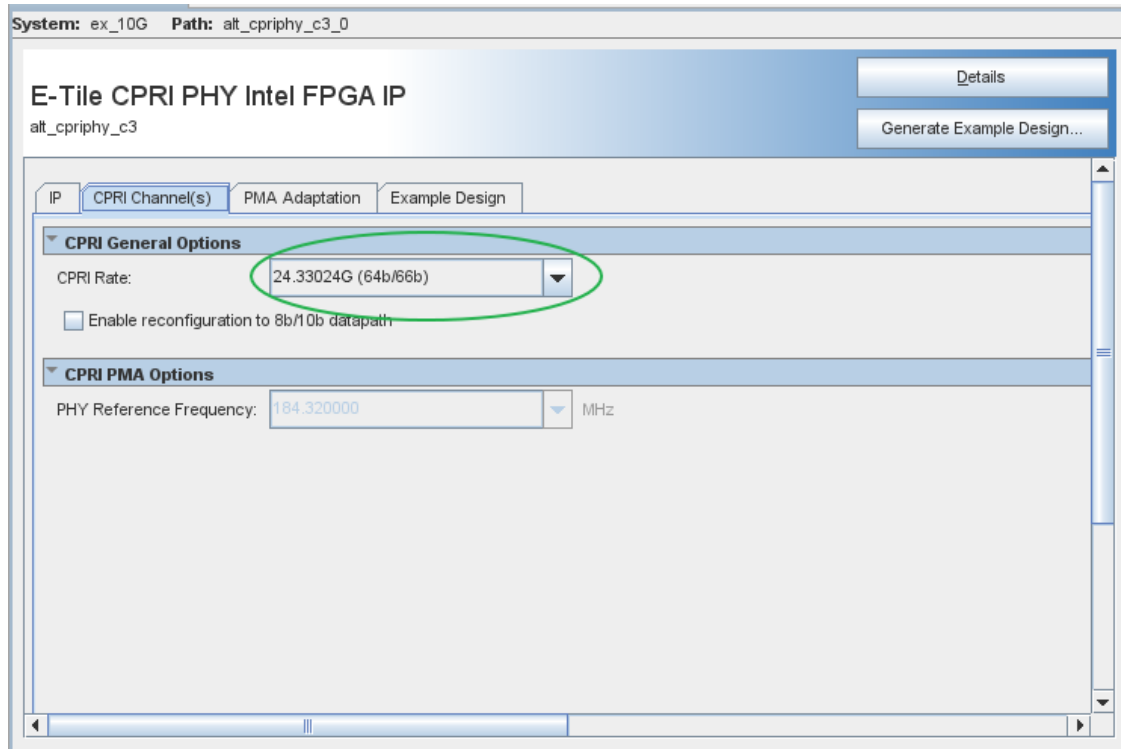


Figure 2-26 Core Variant Setup for CPRI IP

Four transceiver channels are used in the Quartus Project. The pre-compiled .sof files on the demo_batch folder.

■ Demonstration Setup

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

7. Insert a QSFP28 loopback fixture into the QSFP28 port on the Apollo-Som board, as shown in **Figure 2-27**.
8. 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.
9. Goto demo_batch folder and execute test.bat to configure FPGA. There are four pre-compiled sof files are available.

10. Open alt_cpriphy_c3_hw Quartus Project and launch the System Console by selecting the menu item **Tools → System Debugging Tools → System Console** in Quartus.
11. In the System Console window, input the following commands to start the loopback test, as shown in **Figure 2-28**.

```
%cd hwtest_sl  
%source main_script.tcl
```
12. The loopback test report will be displayed in the Tcl Console, as shown in **Figure 2-29**.

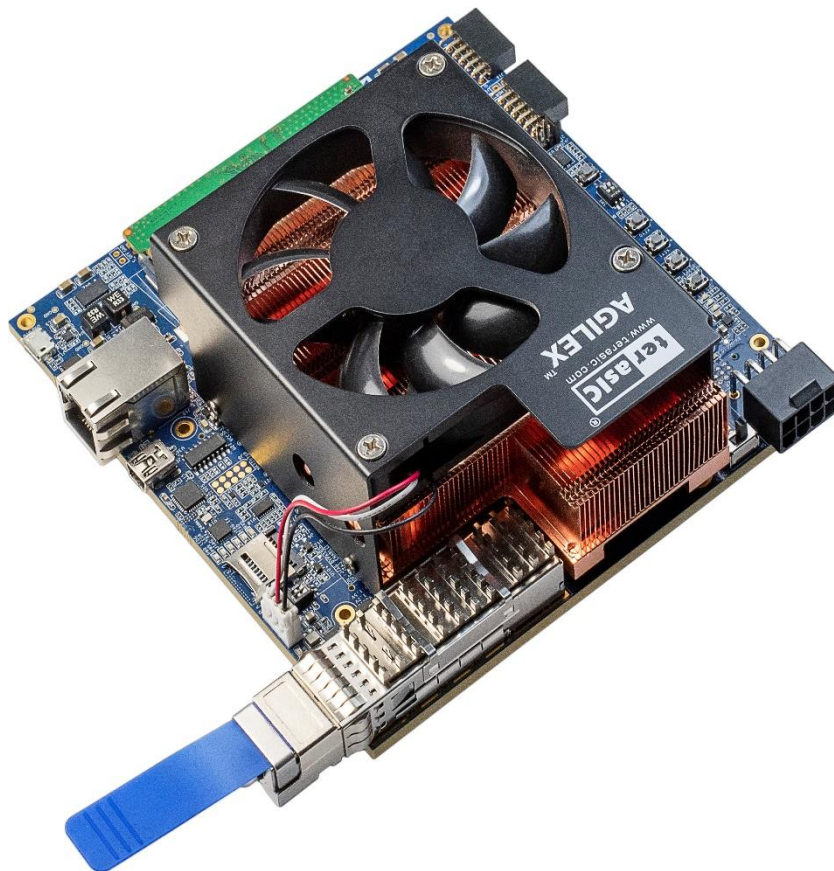


Figure 2-27 Setup QSFP28 loopback fixture

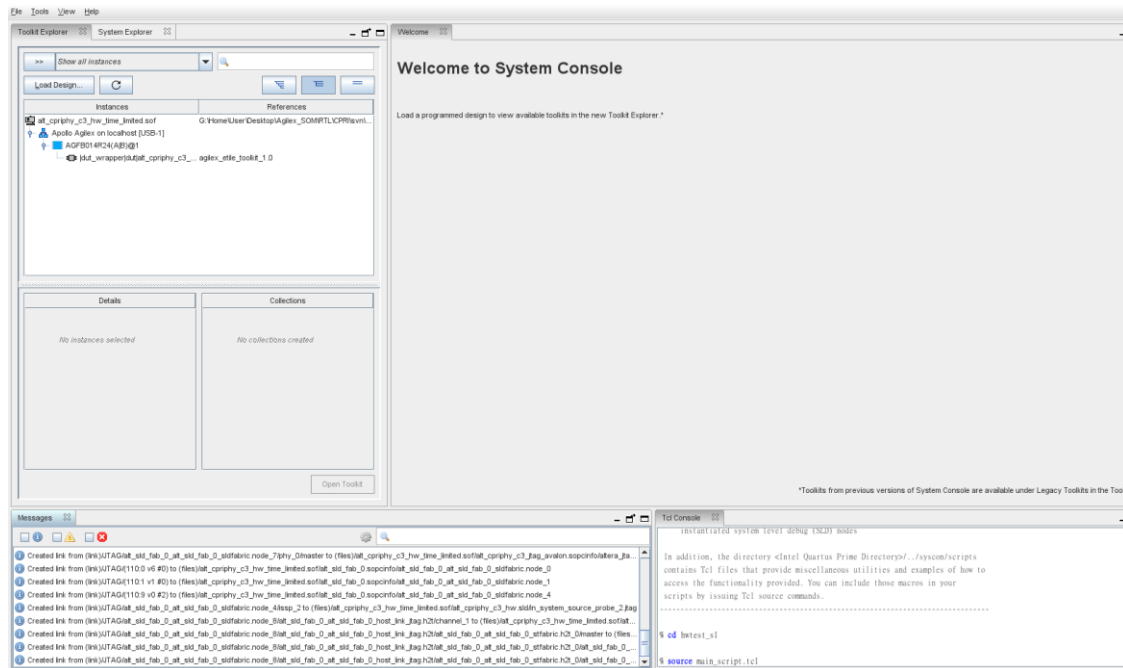


Figure 2-28 Launch the System Console for 24G CPRI Demo

```

Channel 0 : Wait for measure_valid to assert
measure_valid is asserted
Channel 0 : Get checker_pass status:
Checker value = 1
Checker status = Passed!
Channel 0 : Read Deterministic latency counts
Channel 1 : Wait for measure_valid to assert
measure_valid is asserted
Channel 1 : Get checker_pass status:
Checker value = 1
Checker status = Passed!
Channel 1 : Read Deterministic latency counts
Channel 2 : Wait for measure_valid to assert
measure_valid is asserted
Channel 2 : Get checker_pass status:
Checker value = 1
Checker status = Passed!
Channel 2 : Read Deterministic latency counts
Channel 3 : Wait for measure_valid to assert
measure_valid is asserted
Channel 3 : Get checker_pass status:
Checker value = 1
Checker status = Passed!
Channel 3 : Read Deterministic latency counts
Info: Loop 0 passed
End of loop 0

Info: End of c3_cpri_test

Info: Total loop passed = 1/1

Info: Test <c3_cpri_test> Passed

```

Figure 2-29 24G CPRI loopback test report

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 Apollo Agilex board such as users LED/KEY, Network Communication. All the associated files can be found in the directory Demonstrations/SOC of the Apollo Agilex System CD.

To install the demonstrations on the Host computer: Copy the directory Demonstrations into a local directory of your choice. Intel SoC EDS Pro v19.4 is required for users to compile the c-code project.

3.1 HPS 2x6 GPIO Header

This demonstration shows how to use the Linux BSP built-in GPIO driver to control the GPIO port in the GPIO header (J16) to perform loopback test. Note, the Apollo Agilex Module Linux BSP already build-in the GPIO driver.

■ How to control GPIO

Here is an example procedure to control a GPIO N:

1. Export GPIO: Open device file `"/sys/class/gpio/export"`, write a gpio number N to the file, and close the file.
2. Configure GPIO Direction: Open device file `"/sys/class/gpio/gpioN/direction"`, write `"out"` or `"in"` to the file, and close the file.
3. Read/Write GPIO Value: Open device file `"/sys/class/gpio/gpioN/value"`, read/write value to the file, and close the file.
4. Unexport GPIO: Open device file `"/sys/class/gpio/unexport"`, write number N to the file, and close the file.

■ Function Block Diagram

Figure 3-1 shows the function block diagram of the HPS TMD 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.

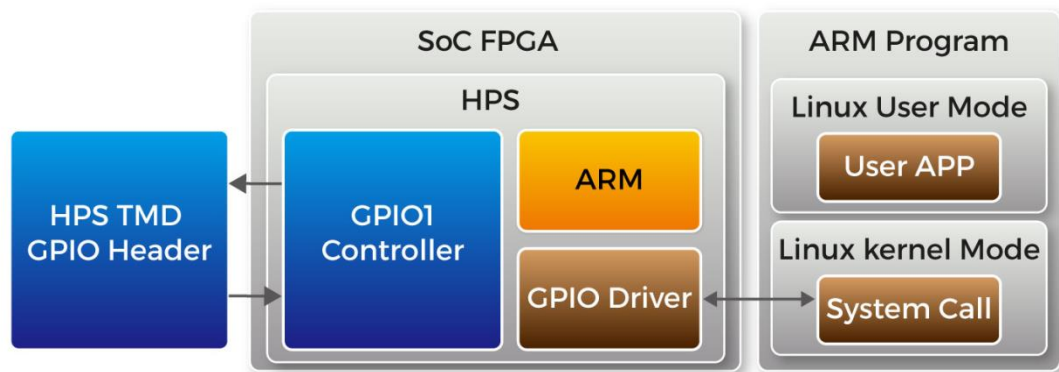


Figure 3-1 Function block diagram of HPS TMD GPIO Header demonstration

■ Function Implement

The c project include main.c and gpio_lib.c files. The main.c implements the loopback test. The gpio_lib.c implement five GPIO 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.

■ Loopback Implement

There are four gpio ports used to loopback. They are HPS_GPIO0, HPS_GPIO1, HPS_GPIO2 and HPS_GPIO3. The Loopback includes two test patterns, the differences between them are data direction and test data value. In test one, we set the four GPIO port 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 gpio's data direction
- Data write and read back
- Verify the received data

■ Demonstration Setup

1. Use two jumper caps to connect HPS_GPIO0 to HPS_GPIO1 and HPS_GPIO2 to HPS_GPIO3 in hps gpio header(J16) on the Apollo Agilex Module. **Figure 3-2** shows the pin location below.

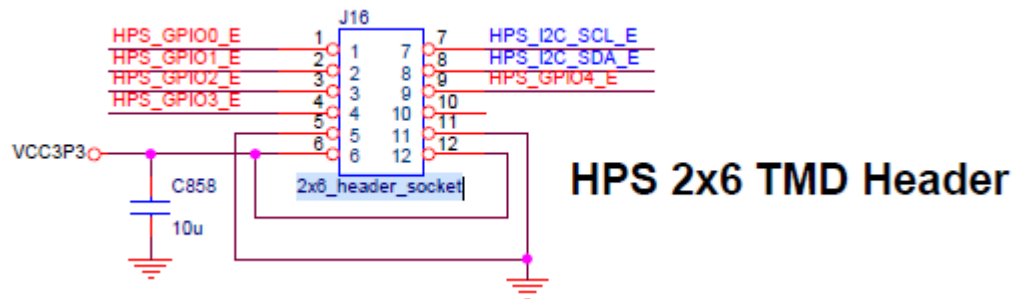


Figure 3-2 GPIO Header Pin location

2. Connect a USB cable to the Mini USB connector (J8) on the Apollo S10 Module and the Host PC.
3. Copy the executable file "**hps_gpo_loopback**" into the microSD card under the **"/home/terasic"** folder in Linux. (Apollo Agilex Module Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Apollo Module Module Linux BSP micro SD card into the board.
5. Power on the Apollo Module Module.
6. Launch Putty to establish the connection between the UART port of Apollo Module Module and the Host PC.
7. In the Putty UART terminal, type user name "terasic" and password "123" to login Linux.
8. Type "**sudo ./hps_gpio_loopback**" in the UART terminal to start the program. Input password "123" if system query password for terasic.
9. You will see the loopback test successfully in the Putty UART terminal as shown in **Figure 3-3**.

```
terasic@localhost:~$ sudo ./hps_gpio_loopback
[sudo] password for terasic:

=====Loopback Test:Start Test1=====
Test 1 : hps_gpio_0->hps_gpio_1,hps_gpio_2->hps_gpio_3, write data: 0x1234f0f0
Recv Data : hps_gpio_1 = 0x0
Recv Data : hps_gpio_3 = 0x1234f0f0
Test1 hps_gpio_0->hps_gpio_1 failed
Test1 hps_gpio_2->hps_gpio_3 successfully

=====Loopback Test:Start Test2=====
Test 2 : hps_gpio_0<-hps_gpio_1,hps_gpio_2<-hps_gpio_3, write data: 0x43210f0f
Recv Data : hps_gpio_0 = 0x0
Recv Data : hps_gpio_2 = 0x43210f0f
Test2 hps_gpio_0<-hps_gpio_1 failed
Test2 hps_gpio_2<-hps_gpio_3 successfully
terasic@localhost:~$
```

Figure 3-3 Loopback test successfully (neet update)

3.2 HPS LED/KEY

This demonstration shows how to use the system call with built-in LED and GPIO driver to control the LED and KEY which are connected to HPS GPIO ports. The built-in GPIO driver is included the Apollo Agilex Module Linux BSP.

■ How to control LED

Here is an example procedure to control the HPS LED:

1. Open LED device: Open device file “/sys/class/leds/hps_led0/brightness”.
2. Turn on/off LED: Write data to the device file for LED control. Write “1” to turn on LED, write “0” to turn off LED.
3. Close LED device: Close the device file.

■ Function Block Diagram

Figure 3-4 shows the function block diagram of the HPS LED/KEY demonstration. The built-in LED and GPIO driver offers interfaces, to which the application can use system call such as open, read, write to access.

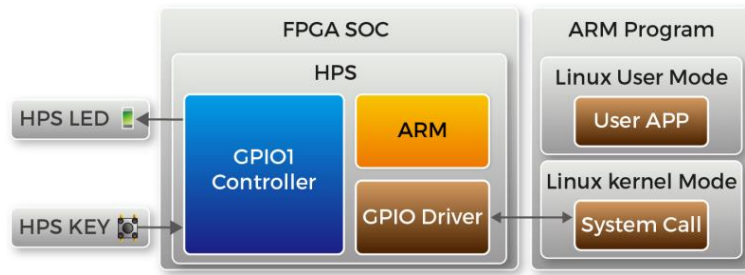


Figure 3-4 Function block diagram of HPS LED/KEY demonstration

■ Function Implement

The c project include main.c, gpio_lib.c and led_lib.c files. The main.c implements the demo main flow. The gpio_lib.c is the same as the one used in HPS TMD GPIO Loobpack Demo. The led_lib implement three LED functions, described as following:

int led_fd_open (unsigned int led);

The led_fd_open function is used to open the LED device file with the specified LED number as parameter. The function return a file descriptor for the LED device.

int led_fd_write (int fd, const void *buf, size_t count);

The led_fd_write function is used to write data to the LED device file. It is used to turn on/off the LED.

int led_fd_close(int fd);

The led_fd_close function is used to close a file descriptor.

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

With the file descriptor return by led_fd_open function, user can use led_fd_write to trun on/off the LED. Call “led_fd_write(fd_led, “1”, 2) “ will turn on the LED, and Call “led_fd_write(fd_led, “0”, 2) “ will turn off the LED

■ Flow Control Implement

The flow control is implemented in main.c. When HPS KEY is pressed, the HPS LED will be turn off. When HPS KEY is released, the HPS LED will be turn on. The GPIO functions implemented in gpio_lib.c are used to monitor HPS KEY status. The LED functions implemented in led_lib.c is used to turn on/off the HPS LED.

Figure 3-5 shows the procedure in main.c file, you can find it's very clear.

```

// export gpio
gpio_export(io_key);
gpio_set_dir(io_key, 0);
↓
fd_led = led_fd_open(io_led);

↓
while (i >= 0) {
    gpio_get_value(io_key, &value);
    if (value)
        led_fd_write(fd_led, "1", 2);
    else
        led_fd_write(fd_led, "0", 2);
    printf("key: %x\n", value);
    sleep(1);
}
↓
led_fd_close(fd_led);
gpio_unexport(io_key);

```

Figure 3-5 LED/KEY implemented in c code

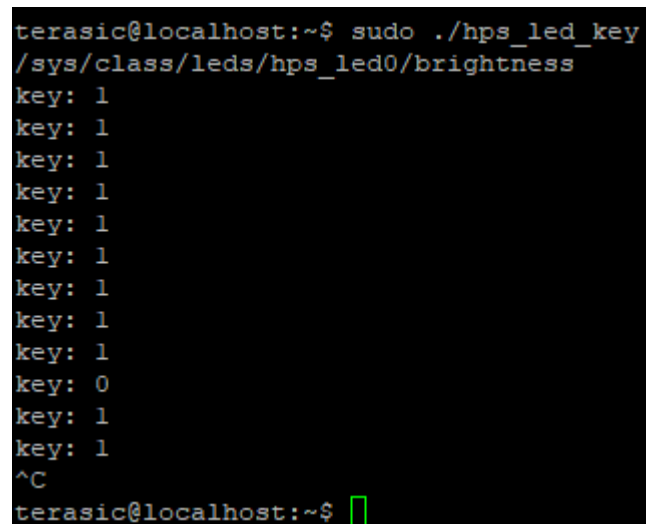
■ Demonstration Source Code

- Build tool: WSL + GNU Compiler
- Project directory: \Demonstration\SoC\hps_led_key
- Binary file: hps_led_key
- Build command: make ('make clean' to remove all temporal files)
- Execute command: sudo ./hps_led_key

■ Demonstration Setup

1. Connect a USB cable to the Mini USB connector (J8) on the Apollo S10 Module and the Host PC.
2. Copy the executable file "**hps_led_key**" into the microSD card under the "**/home/terasic**" folder in Linux. (Apollo Agilex Module Linux BSP has pre-installed this code, so users can skip this copy action.)
3. Insert the Apollo Agilex Module Linux BSP micro SD card into the Apollo Agilex Module.
4. Power on the Apollo Agilex Module.
5. Launch Putty to establish the connection between the UART port of Apollo S10 Module and the Host PC.
6. In the Putty UART terminal, type user name "terasic" and password "123" to login Linux.
7. Type "**sudo ./hps_led_key**" in the UART terminal to start the program. Input password "123" if system query password for terasic.

8. You will see the loopback test successfully in the Putty UART terminal as shown in **Figure 3-6**.
9. Press HPS KEY will make key value become 0 and HPS LED unlighten.
10. Press CTRL+C can terminate the program.



```
terasic@localhost:~$ sudo ./hps_led_key
/sys/class/leds/hps_led0/brightness
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 1
key: 0
key: 1
key: 1
^C
terasic@localhost:~$
```

Figure 3-6 LED/KEY test

3.3 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 3-7**. 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.

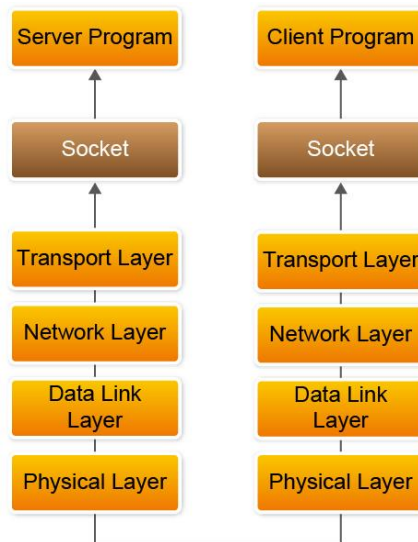


Figure 3-7 Communicate on a network via a socket

■ 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 3-8** shows the communication diagram between the client and server.

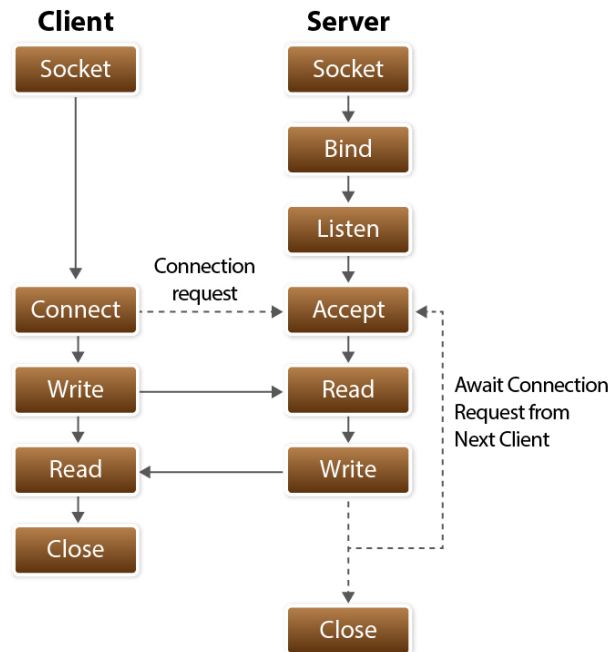


Figure 3-8 Client and Server communication

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

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 3-9** 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 3-9 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 3-10** 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 IPv4v 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 3-10 Socket Client Code

■ Demonstration Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 3-11**. 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.

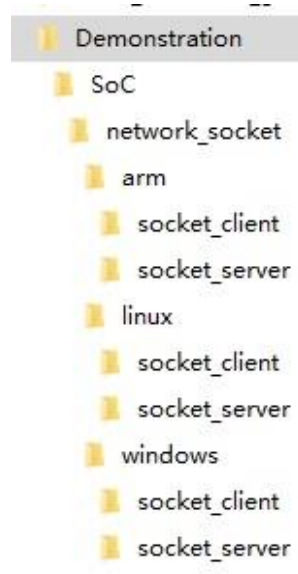


Figure 3-11 Source Code Folder Tree

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 Apollo Agilex Module to Network via Ethernet port (J10).
2. Connect a USB cable to the Mini USB connector (J9) on the Apollo Agilex Module and the Host Windows PC.
3. Copy the executable file "**socket_server**" into the microSD card under the `"/home/terasic"` folder in Linux. (Apollo Agilex Module Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Apollo Agilex Module Linux BSP micro SD card into the Apollo S10 Module.
5. Power on the Apollo Agilex Module.
6. In Windows, launch the Putty to connect Apollo Agilex Module via the USB-to-UART link.
7. In the Putty, type user name "**terasic**" and password "**123**" to login Linux.
8. Type "**ifconfig**" to query the IP address which will be used in `socket_client`.
9. Type "**./socket_server 2020**" to launch the server program with port number 2020 as shown in [Figure 3-12](#). The port number can be any value between 2000 and 63500.

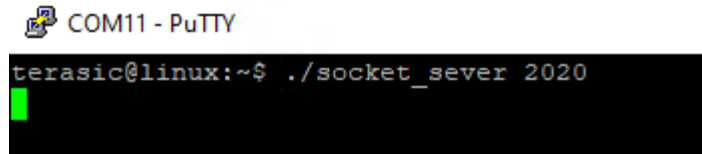


Figure 3-12 Start Socket Server

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

1. Make sure the WSL is installed on your Windows and the Windows is connected to a network.
2. Launch WSL.
3. Copy the client program (linux/socket_client/socket_client) in the example kit to the WSL.
4. In the WSL, change the current directory to the directory where socket_client is located.
5. Then, type “./socket_client <ip address> 2020” to launch the client program to connect to the Host server with port number 2020 as shown in **Figure 3-13**.



Figure 3-13 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 3-14**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 3-15**.

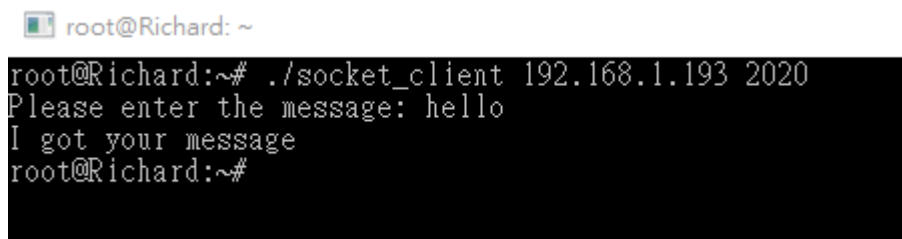
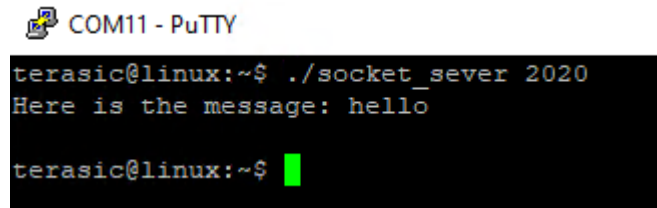


Figure 3-14 Send Message in Client Program



```
COM11 - PuTTY
terasic@linux:~$ ./socket_sever 2020
Here is the message: hello
terasic@linux:~$
```

Figure 3-15 Server dumps received message

3.4 Build C/C++ Project

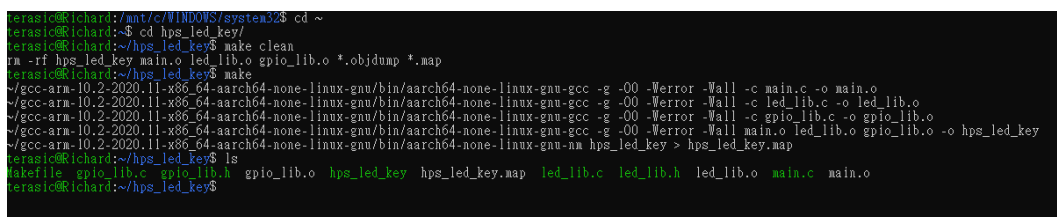
This section describes how to recompile the above C/C++ project included in the System CD.

First, user need to install tool chain:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. Type
“wget https://developer.arm.com/-/media/Files/downloads/gnu-a/10.2-2020.11/binrel/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu.tar.xz”
4. Type “tar xf gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu.tar.xz”
5. Type “git clone <https://github.com/altera-opensource/intel-socfpga-hwlib>” to download HPS hardware library.

Here is the procedure to compile the example projects in System CD:

1. Login Linux or WSL on Windows.
2. Type “cd ~”
3. Copy the CD Demo project into the Linux System and go to the project folder.
4. Type “make” to build project as shown in **Figure 3-16**.



```
terasic@Richard:/mnt/c/WINDOWS/system32$ cd ~
terasic@Richard:~$ cd hps_led_key/
terasic@Richard:~/hps_led_key$ make clean
rm -rf hps_led_key main.o led_lib.o gpio_lib.o *.objdump *.map
terasic@Richard:~/hps_led_key$ make
~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c main.c -o main.o
~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c led_lib.c -o led_lib.o
~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall -c gpio_lib.c -o gpio_lib.o
~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc -g -O0 -Werror -Wall main.o led_lib.o gpio_lib.o -o hps_led_key
~/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-nm hps_led_key > hps_led_key.map
terasic@Richard:~/hps_led_key$ ls
Makefile gpio_lib.c gpio_lib.h gpio_lib.o hps_led_key hps_led_key.map led_lib.c led_lib.h led_lib.o main.c main.o
terasic@Richard:~/hps_led_key$
```

Figure 3-16 Build C/C++ Project

Chapter 4

Additional Information

4.1 Getting Help

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

■ Terasic Technologies

No.80, Fenggong Rd., Hukou Township, Hsinchu County 303035. Taiwan

Email: support@terasic.com

Web: www.terasic.com

Apollo Agilex Web: agilex-som.terasic.com

■ Revision History

Date	Version	Changes
2022.07	First publication	
2022.08	V1.1	Modify chapter 3