Apollo S10 SOM

Essential Instrument of
High Performance Computing

User Manual

Contents

Chapter 1

Overview

This Manual will introduce the various application demonstrations on Apollo S10 SoM board. These demonstrations cover most of the interfaces on Apollo S10 SoM. Let users familiarize using these interfaces of the Apollo S10 SoM 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 19.4.0** or later.
- **Intel SoC Embedded Design Suite(EDS) Professional Edition**

Chapter 2

Examples For FPGA

This chapter provides examples of advanced designs implemented by RTL or Qsys on the Apollo S10 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 S10 SoM System CD.

2.1   Configure Si5341A in RTL

There are two Silicon Labs Si5341A clock generators on Apollo S10 FPGA board can provide adjustable frequency reference clock (See **Figure 2-1**) for FMC/FMC+ connectors and DDR4 memory. The Si5341A clock generator can output four differential frequencies from 100Hz ~ 712.5Mhz though I2C interface configuration. This section will show you how to use FPGA RTL IP to configure each Si5341A PLL and generate users desired output frequency to each peripheral.

Apollo S10 SoM
Demonstration
Manual

5

www.terasic.com
October 14,
2020
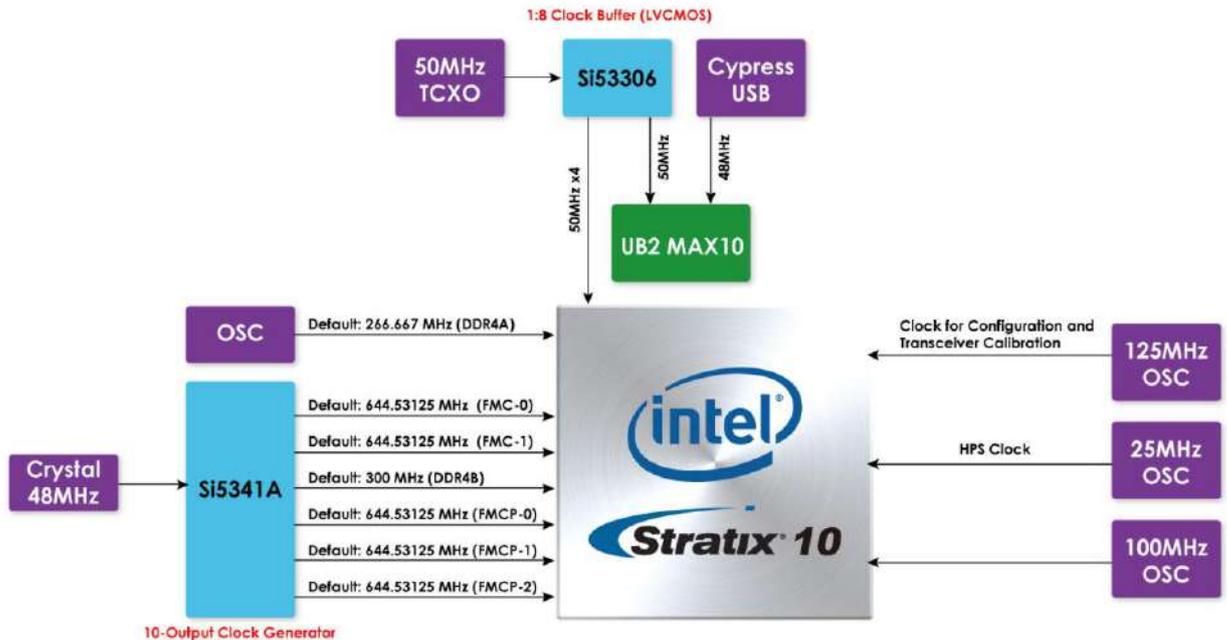
Figure 2-1 Clock tree of the Apollo S10 SoM

■ **Creating Si5341A Control IP**

The Si5341A control IP is located in the folder: "\Demonstrations\FPGA\Si5341A_IP" 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.

Also, System Builder tool (located in System CD) can be used to help developer to set Si5341A to output desired frequencies, and generate a Quartus project with control IP. In the System Builder window, users can select desired frequencies by selecting a desired output frequency in the pull down menu as shown in **Figure 2-2**. For details about the System Builder, please refer to Chapter 3 – System Builder in the Apollo S10 SoM user manual.

Apollo S10 SoM
Demonstration
Manual

6

www.terasic.com
October 14,
2020
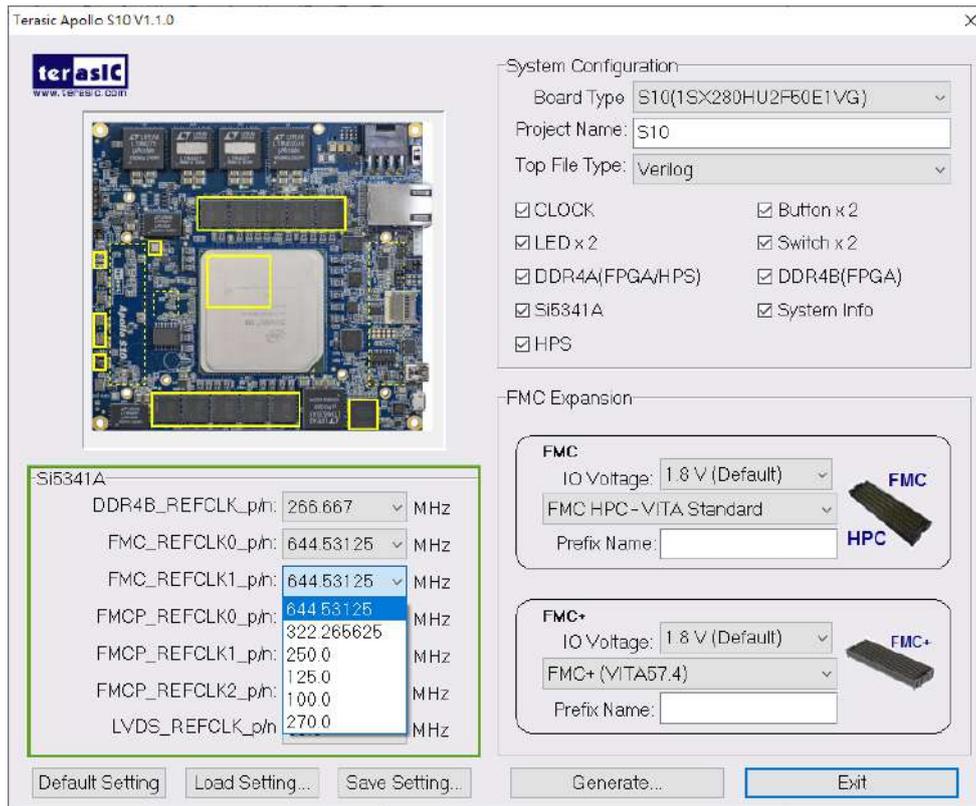
Figure 2-2 Select Desired Si5341A output frequency

■ **Using Si5341 control IP**

Table 2-1 lists the instruction ports of Si5341A Controller IP.

Table 2-1 Si5341A 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）
iDDR4B_REFCLK iFMC_REFCLK0 iFMC_REFCLK1 iFMCP_REFCLK0 iFMCP_REFCLK1 iFMCP_REFCLK2	input	Setting Si5341A Output Channel Frequency Value

Apollo S10 SoM
Demonstration
Manual

www.terasic.com

October 14,
2020

iLVDS_REFCLK		
oPLL_REG_CONFIG_DONE	output	Si5341A Configuration status (0: Configuration in Progress, 1: Configuration Complete)
I2C_DATA	inout	I2C Serial Data to/fromSi5341A
I2C_CLK	output	I2C Serial Clock to Si5341A

As shown in **Table 2-2** and **Table 2-3**, both two Si5341A control IP have preset several output frequency parameters, if users want to change frequency, users can fill in the input ports "iDDR4B_REFCLK ", "iFMC_REFCLK0", "iFMC_REFCLK1", " iFMCP_REFCLK0", "iFMCP_REFCLK1", "iFMCP_REFCLK2", and " iLVDS_REFCLK" with desired frequency values and recompile the project. For example, in the components Si5341A1, change

. iFMC_REFCLK0 (`XCVR_REF_644M53125),
to. iFMC_REFCLK0 (`XCVR_REF_322M265625),

Recompile project, the Si5341A OUT0 channel (for FMC) output frequency will change from 644.53125Mhz to 322.26562Mhz.

Table 2-2 Si5341A Controller Reference Clock Frequency Setting for FMC/FMC+

iFMC_REFCLK0/1 iFMCP_REFCLK0/1/2 Input Setting	Si5341A Channel Clock Frequency(MHz)
4'h0	644.53125
4'h1	322.265625
4'h2	250
4'h3	125
4'h4	100
4'h5	270

Table 2-3 Si5341A Controller Reference Clock Frequency Setting for DDR4B

iDDR4B_REFCLK Input Setting	Si5341A Channel Clock Frequency(MHz)
4'h0	300

4'h1	266.667
4'h2	233.333
4'h3	166.667

Users can also dynamically modify the input parameters, and input a positive edge trigger for "iStart", then, Si5341A 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 Si5341A 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 2-3**), which is provided by Silicon Labs. This tool can help users to set the Si5341A'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_oftware
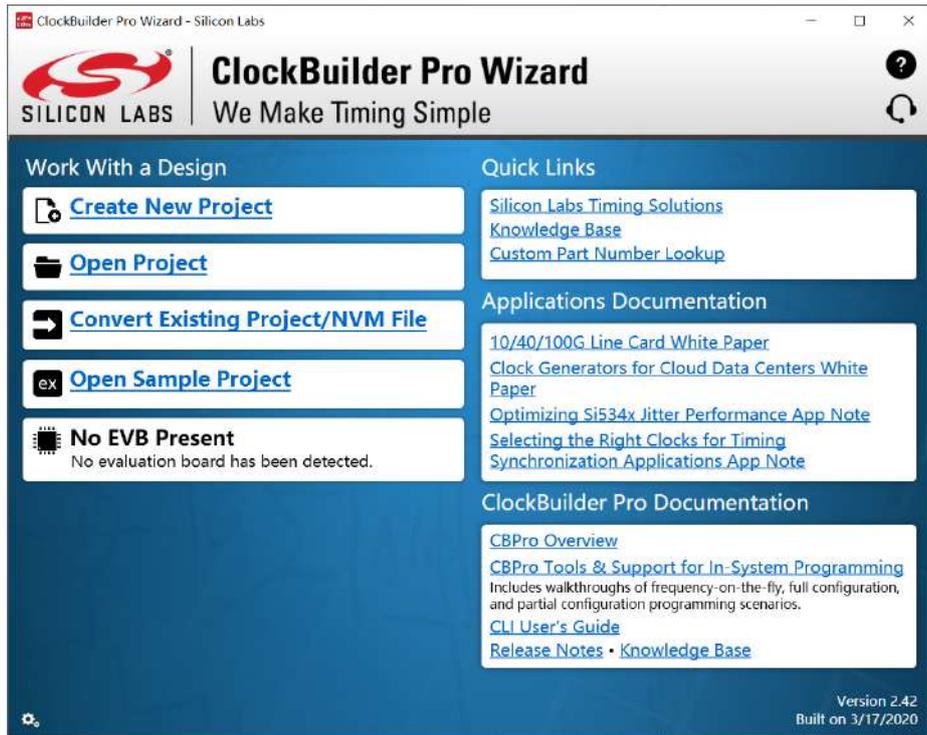
Figure 2-3 ClockBuilder Pro Wizard

2. After the installation, select Si5341, and configure the input frequency and output frequency as shown in **Figure 2-4**.

terasIC
www.terasic.com

Apollo S10 SoM
Demonstration
Manual

10

www.terasic.com

October 14,
2020
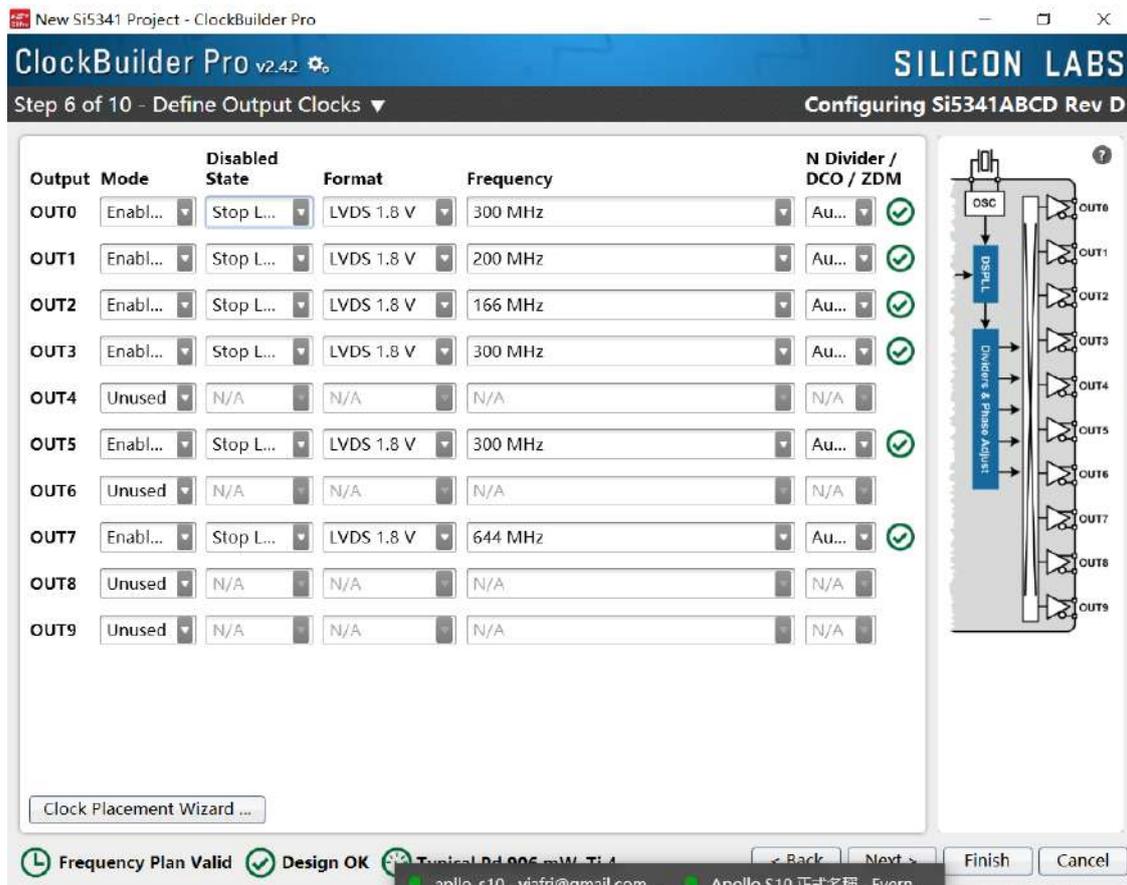
Figure 2-4 Define Output Clock Frequencies on ClockBuilder Pro Wizard

3. After the setting is completed, ClockBuilder Pro Wizard generates a Design Report(text), which contains users setting frequency corresponding register value (See **Figure 2-5**).
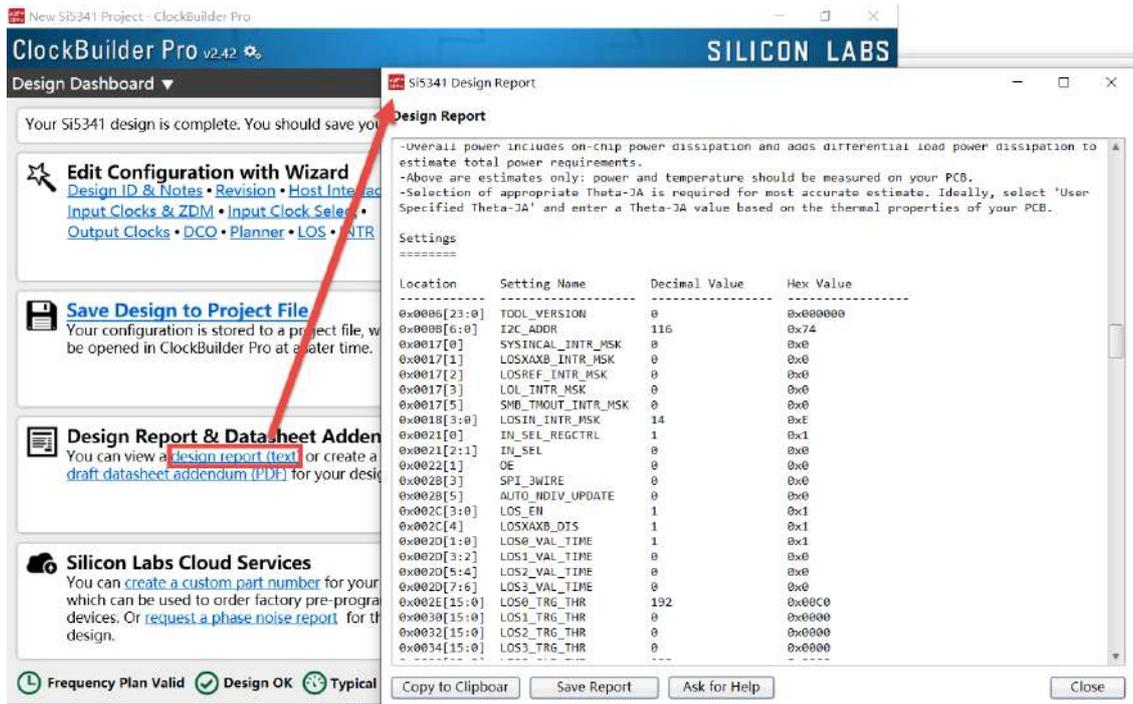
Figure 2-5 Open Design Report on ClockBuilder Pro Wizard

4. Open Si5341 control IP sub-module "si5341a_i2c_reg_controller.v " as shown in **Figure 2-6**, refer to Design Report parameter to modify sub-module corresponding register value (See **Figure 2-7**).
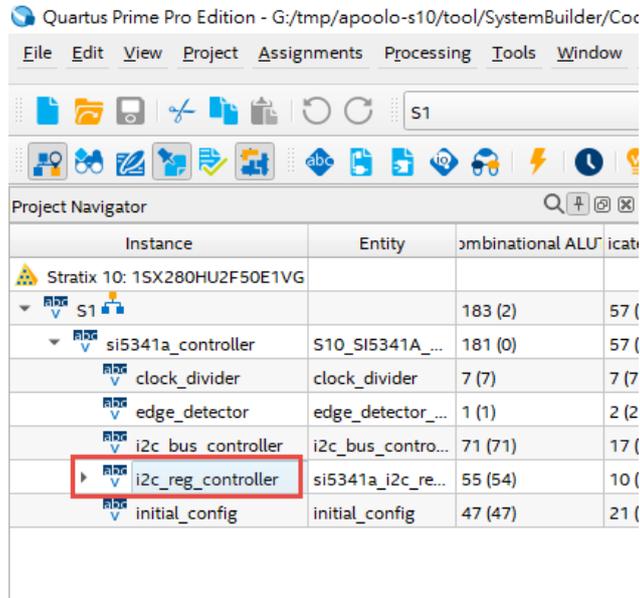

Figure 2-6 Sub-Module file "Si5341A_i2c_reg_controller.v"

Apollo S10 SoM
Demonstration
Manual

www.terasic.com

October 14,
2020
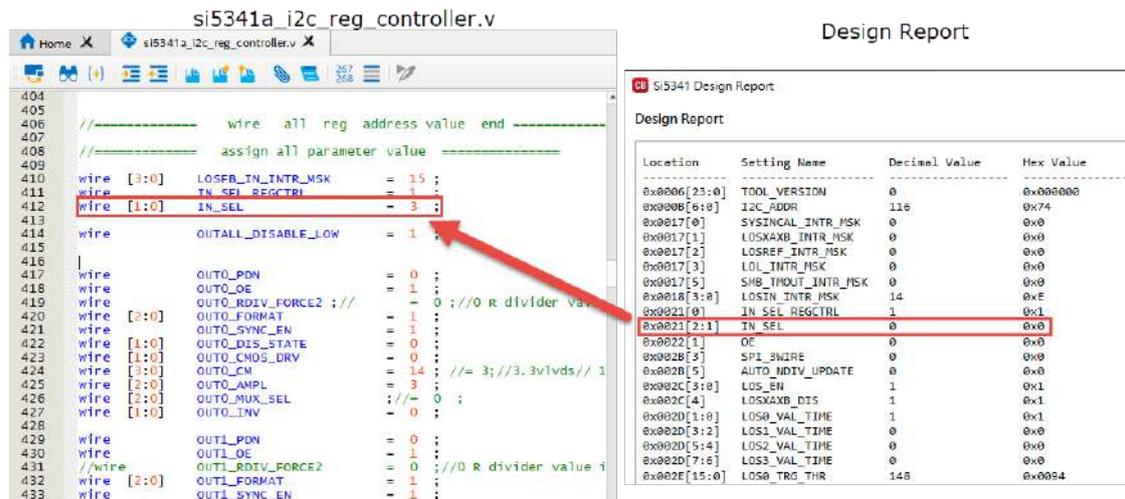
Figure 2-7 Modify Si5341A Control IP Base on Design Report

After modifying and compiling, Si5341A can output new frequencies according to the users' setting.

Note :

No need to modify all Design Report parameters in si5341a_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 SI5341A/ Temperature/ Power/ Fan

This demonstration shows how to use the Nios II processor to program programmable clock generators (Si5341A) 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 2-8 shows the system block diagram of this demonstration. The Si5341A 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

Apollo S10 SoM
Demonstration
Manual

www.terasic.com
October 14,
2020

and controlled by internal logic circuits. All collected status data or control commands will be sent to the SPI slave block so that the Stratix 10 FPGA can read it through the SPI interface.

In the Stratix 10 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.
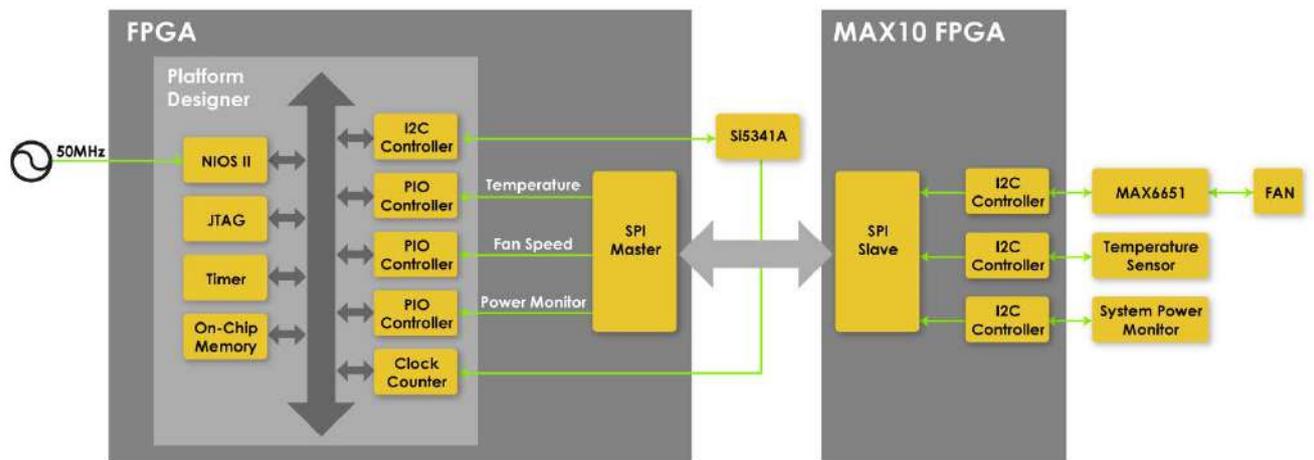

Figure 2-8 Block Diagram of the Nios II Basic Demonstration

The system provides a menu in nios-terminal, as shown in **Figure 2-9** 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.

Figure 2-9 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 R4 (0.003 Ω) for LTC2945 in the circuit, when power on the Apollo S10, there will be a voltage drop (named ∆SENSE Voltage) on R4. 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 TERASIC 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 Si5341A programming, please note the device I2C address is 0xEE. The program can control the Si5341A to configure the output frequency of FMC_REFCLK0, FMC_REFCLK1, FMCP_REFCLK0, FMCP_REFCLK1,

Apollo S10 SoM Demonstration Manual

www.terasic.com

October 14, 2020

FMCP_REFCLK2, DDR4B_REFCLK and LVDS_REFCLK according to your choice.

■ **Demonstration File Location**

● Hardware project directory: NIOS_BASIC_DEMO

● Bitstream used: S10_top.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 Si5341A test, please input key '0' and input the desired output frequency for eight clock sources, as shown in **Figure 2-10**.

![terasic logo](www.terasic.com)

Apollo S10 SoM
Demonstration
Manual

16

www.terasic.com
October 14,
2020

```
Altera Nios II EDS 19.1 [gcc4]                                              —    □    ×
======= Si5341A Programming =======
[0] 644.531250 MHz
[1] 322.265625 MHz
[2] 250.000000 MHz
[3] 125.000000 MHz
[4] 100.000000 MHz
[5] 270.000000 MHz
[Other] exit
please select FMC_REFCLK0:5
please select FMC_REFCLK1:4
please select FMCP_REFCLK0:3
please select FMCP_REFCLK1:2
please select FMCP_REFCLK2:1
[0] 300.000000 MHz
[1] 266.666992 MHz
[2] 233.332993 MHz
[3] 166.667007 MHz
[Other] exit
please select DDR4B_REFCLK:1
[0] 50.000000 MHz
[1] 100.000000 MHz
[Other] exit
please select LVDS_REFCLK:0

I2C core is enabled!
FMC_REFCLK0/270.000000MHz ref clock test PASS (clk1=998, clk2=5389, expected clk2=5389)
FMC_REFCLK1/100.000000MHz ref clock test PASS (clk1=998, clk2=1995, expected clk2=1996)
FMCP_REFCLK0/125.000000MHz ref clock test PASS (clk1=998, clk2=2495, expected clk2=2495)
FMCP_REFCLK1/250.000000MHz ref clock test PASS (clk1=998, clk2=4990, expected clk2=4990)
FMCP_REFCLK2/322.265625MHz ref clock test PASS (clk1=998, clk2=6430, expected clk2=6432)
DDR4B_REFCLK/266.666992MHz ref clock test PASS (clk1=998, clk2=5323, expected clk2=5322)
LVDS_REFCLK/50.000000MHz ref clock test PASS (clk1=998, clk2=998, expected clk2=998)
Si5341A Test:PASS
======= Stratix 10 Demo Program =======
[0] Si5341A
[1] Display Board Info
Input your chioce:_
```

Figure 2-10 Si5341A Demo

7. For temperature, power monitor and fan test, please input key '1' and press 'Enter' in the nios-terminal, as shown in **Figure 2-11**.

Figure 2-11 Board Info Demo

2.3 DDR4 SDRAM RTL Test

This demonstration performs a memory test function on the DDR4 memory (DDR4A and DDR4B) on the Apollo S10 SoM board. The memory size of each DDR4 bank used in this test is 32GB.

■ **Function Block Diagram**

Figure 2-12 shows the function block diagram of this demonstration. There are two DDR4 SDRAM controllers. All of the controllers (DDR4A and DDR4B) use 266.66 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. 266.66 MHz.
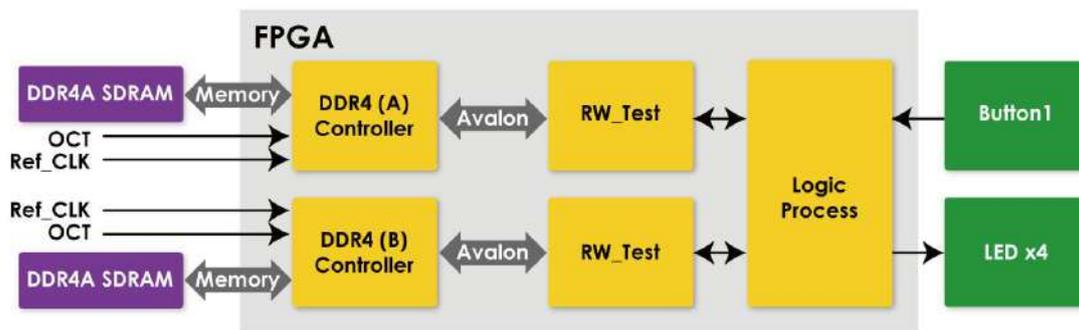
Figure 2-12 Block diagram of the DDR4 RTL demonstration

■ Stratix 10 External Memory Interfaces

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

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

■ Design Tools

- Quartus Prime 19.4 Pro Edition or later

■ Demonstration Source Code

- Project Directory: Demonstration\FPGA\RTL_DDR4_Test
- Bit Stream: S10_top.sof
- Demonstration Batch File : RTL_DDR4_Test\demo_batch

  The demo batch file includes following files:

  ◆ Batch File: test.bat
  ◆ FPGA Configuration File: S10_top.sof

■ Demonstration Setup

1. Make sure Quartus Prime Pro Edition is installed on the Host PC.
2. Connect the Apollo S10 SoM board to the Host PC via the USB cable. Install the USB-Blaster II driver if necessary.

3. Power on the Apollo S10 SoM board.

4. Execute the demo batch file "test.bat" under the batch file folder \RTL_DDR4_Test\demo_batch.

5. Press **KEY1** (see **Figure 2-13**) to start DDR4 write & loopback verify process. It will take about 2~3 second to perform the test. While testing, the LED will blink. When LED stop blinking it means the test process is done. In this case, if the LED light, it means the test result is passed. If the LED is no light, it means the test result is failed. The **LED0** represents the test result for the DDR4A, the **LED1** represents the test result for the DDR4B.

6. Press **KEY1** again to regenerate the test control signals for a repeat test.
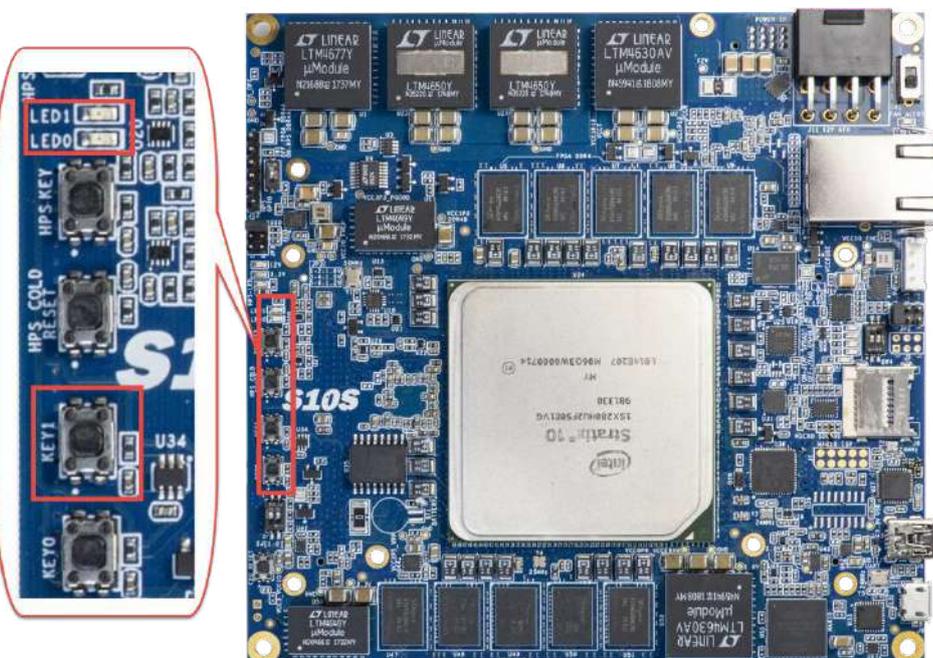

Figure 2-13 Location of the KEY and LED on the Apollo S10 SoM board

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 Stratix 10 External Memory Interfaces is used to access the two DDR4 SDRAM banks on the FPGA board, and how the Nios

Apollo S10 SoM
Demonstration
Manual

www.terasic.com

October 14,
2020

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 2-14 shows the system block diagram of this demonstration. In the Platform Designer (formerly Qsys), one 50 MHz, dual frequency OSC and PLL clock generator(Si5341A) are used. The Si5341A and dual frequency OSC will provide 266.67Mhz clock to the DDR4A and DDR4B bank as the reference clock. There are two DDR4 Controllers which are used in the demonstrations. Each controller is responsible for one DDR4 bank (DDR4A and DDR4B). Each DDR4 controllers are configured as a 32GB DDR4-1066Mhz controller. 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.
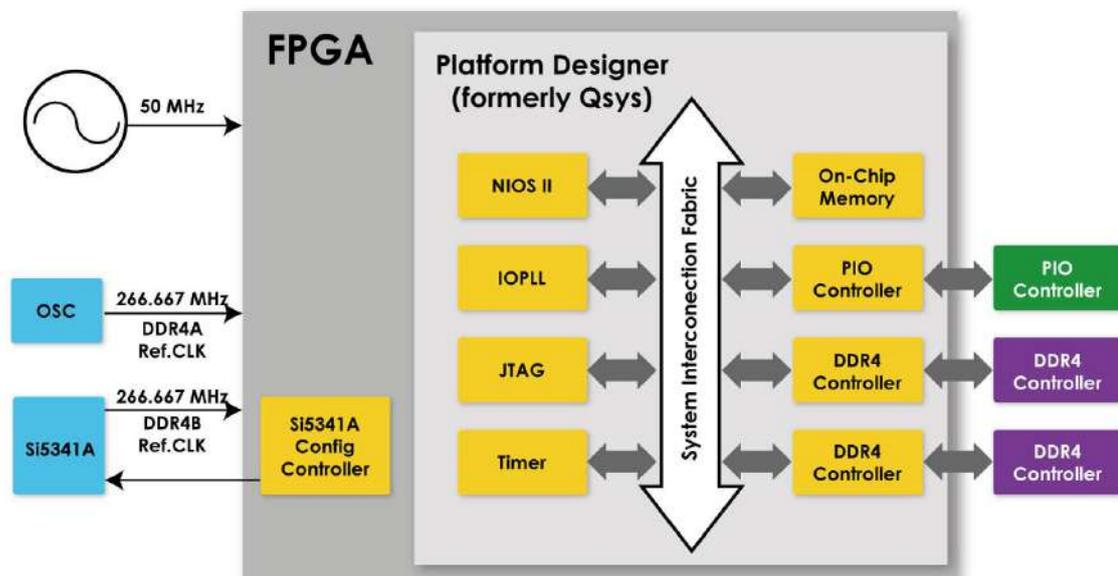

Figure 2-14 Block diagram of the DDR4 Basic Demonstration

The system flow is controlled by a Nios II program. First, the Nios II program writes test patterns into the whole 32GB of SDRAM. Then, it calls Nios II system function,

Apollo S10 SoM
Demonstration
Manual

www.terasic.com
October 14,
2020

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.

■ **Design Tools**

- Quartus Prime 19.4 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: S10_top.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

terasic
www.terasic.com

Apollo S10 SoM
Demonstration
Manual

22

www.terasic.com
October 14,
2020

"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 2-15**. 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 2-16**. 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.

Figure 2-15 Progress option [0] DDR4x2 Test

Figure 2-16 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 Stratix 10 FPGA and allows users to obtain board status information such as power, temperature, and fan speed on the Apollo S10 board.

The Apollo S10 board provides several sensors to monitor the status of the board, such as FPGA temperature, board power monitor, and fan speed status. These interfaces are connected to the system MAX FPGA on the board. The logic in the system MAX FPGA will automatically read the status values of these sensors and store them in the internal register. As shown in **Figure 2-17**, 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.
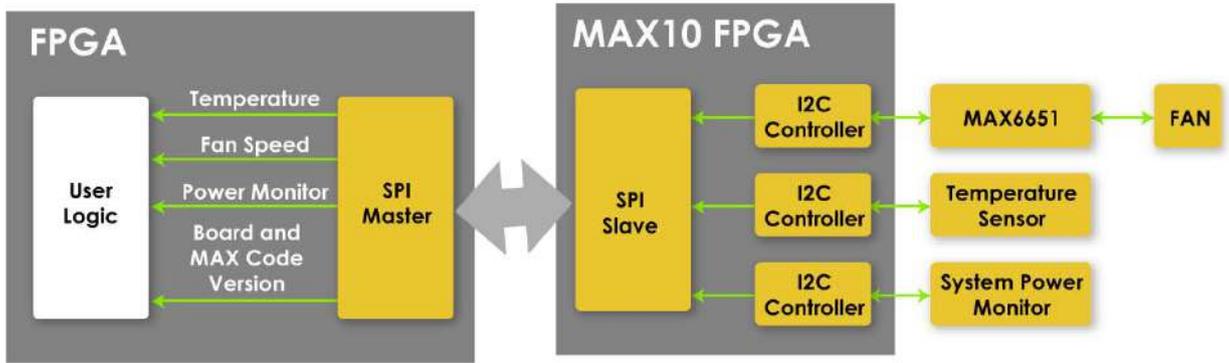
Figure 2-17 the board information IP architecture

User can placing a board information IP (BOARD_INFO.v ; SPI master) provided by Terasic in the Stratix 10 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 2-18 shows the input and output pins of the board information IP. Detailed pin descriptions and functions can be obtained from **Table 2-4** 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 boar 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 2-19** shows the status of the IP during execution.
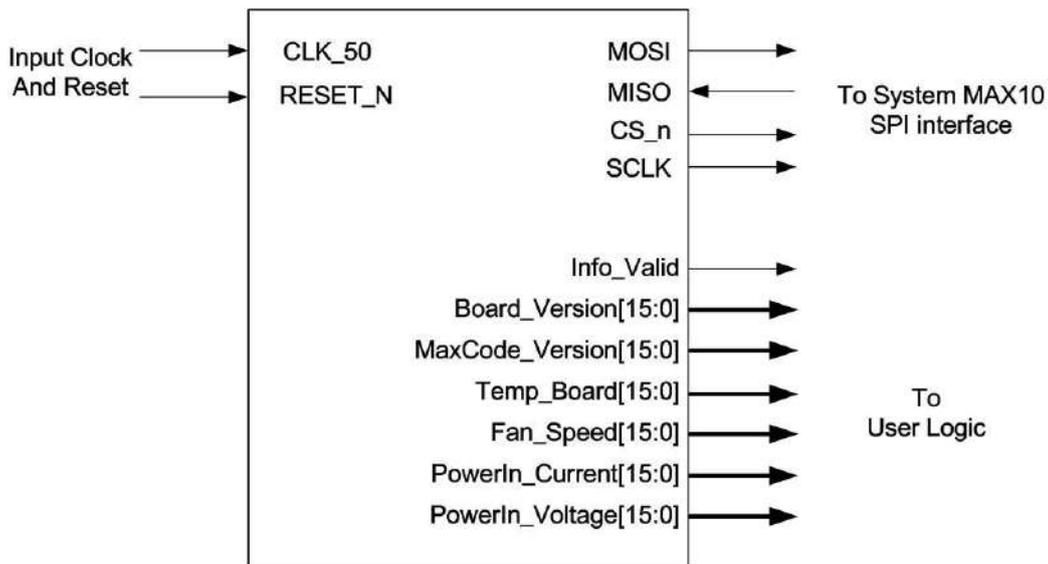
Figure 2-18 Pin out of the board information IP

Table 2-4 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 output. Please connect this signal to the **INFO_SPI_MOSI** pin.
MISO	Input	1	Master 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 salve. 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 Apollo S10 board. It will be started at 0x0001.
MaxCode_Version	Output	16	This information indicates the version of the System MAX 10 FPGA code. It will be started at 0x000A.
Temp_Board	Output	16	Ambient temperature of the development

|			board. The unit of the output value is Celsius.
Temp_FPGA	Output	16	FPGA temperature of the development board. The unit of the output value is Celsius.
Fan_Speed	Output	16	Fan speed of the board. The unit of the output value is RPM.
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** output value is "1816" that means 1.816A for 12V power.
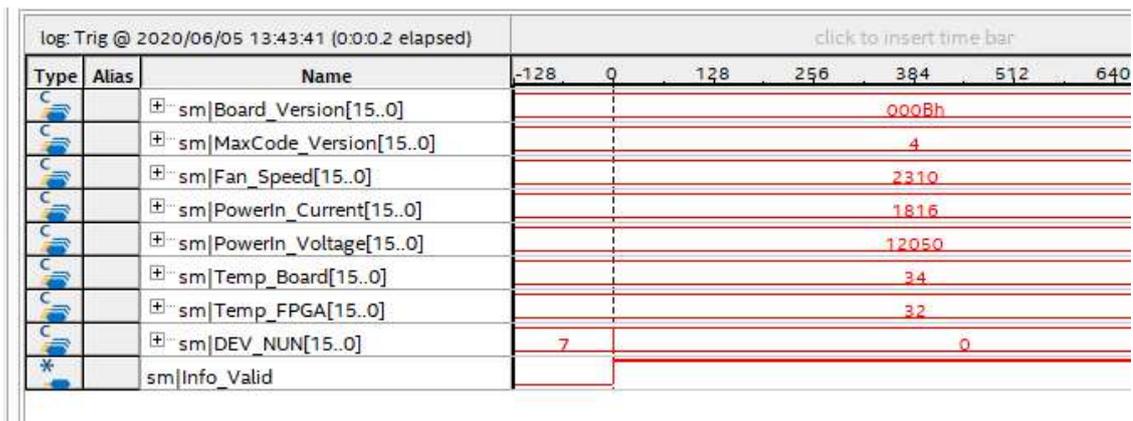
Figure 2-19 Waveform of the board status output

Apollo S10 SoM
Demonstration
Manual

27

www.terasic.com
October 14,
2020

<div align="right">

Chapter 3

</div>

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 S10 SoM board such as users LED/KEY, Network Communication. All the associated files can be found in the directory Demonstrations/SOC of the Apollo S10 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 1x6 GPIO Header

This demonstration shows how to use the Linux BSP built-in GPIO driver to control the GPIO port in the GPIO header (J12) to perform loopback test. Note, the Apollo S10 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.
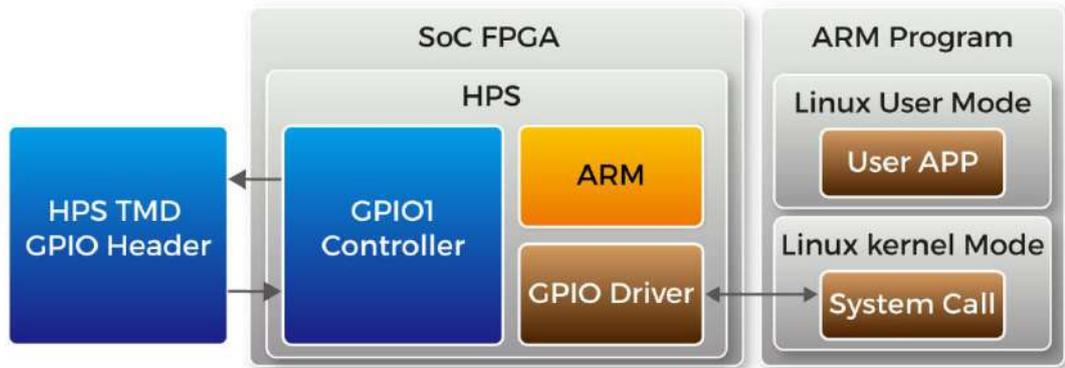
Figure 3-1 Function block diagram of HPS TMD GPIO Header demonstration

■ Function Implement

The c project include main.c and gpio_lib.c files. The main.c implements the loopack 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);

Apollo S10 SoM
Demonstration
Manual

29

www.terasic.com
October 14,
2020

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 wirte "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

Apollo S10 SoM
Demonstration
Manual

www.terasic.com
October 14,
2020

■ Demonstration Setup

1. Use two jumper caps to connect HPS_GPIO0 to HPS_GPIO1 and HPS_GPIO2 to HPS_GPIO3 in hps gpio header(J12) on the Apollo S10 Module. **Figure 3-2** shows the pin location below.
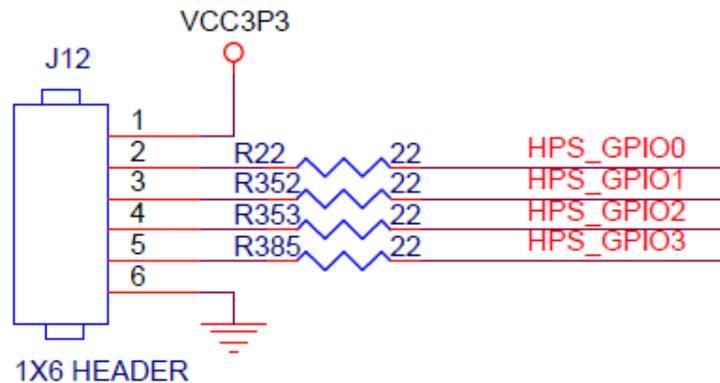

Figure 3-2 GPIO Header Pin location

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 S10 Module Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Apollo S10 Module Linux BSP micro SD card into the board.
5. Power on the Apollo S10 Module.
6. Launch Putty to establish the connection between the UART port of Apollo S10 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**.

Figure 3-3 Loopback test successfully

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 S10 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.

Apollo S10 SoM
Demonstration
Manual

32

www.terasic.com

October 14,
2020
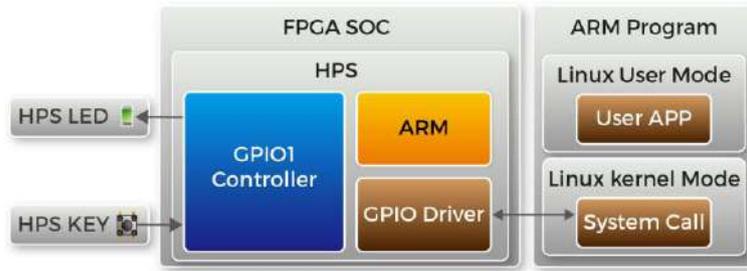
Figure 3-4 Function block diagram of HPS LED/KEY demonstration

■ **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.

Apollo S10 SoM
Demonstration
Manual

33

www.terasic.com
October 14,
2020

```
// 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 S10 Module Linux BSP has pre-installed this code, so users can skip this copy action.)
3. Insert the Apollo S10 Module Linux BSP micro SD card into the Apollo S10 Module.
4. Power on the Apollo S10 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

Apollo S10 SoM
Demonstration
Manual

34

www.terasic.com
October 14,
2020

9. Press CTRL+C can terminate the program.
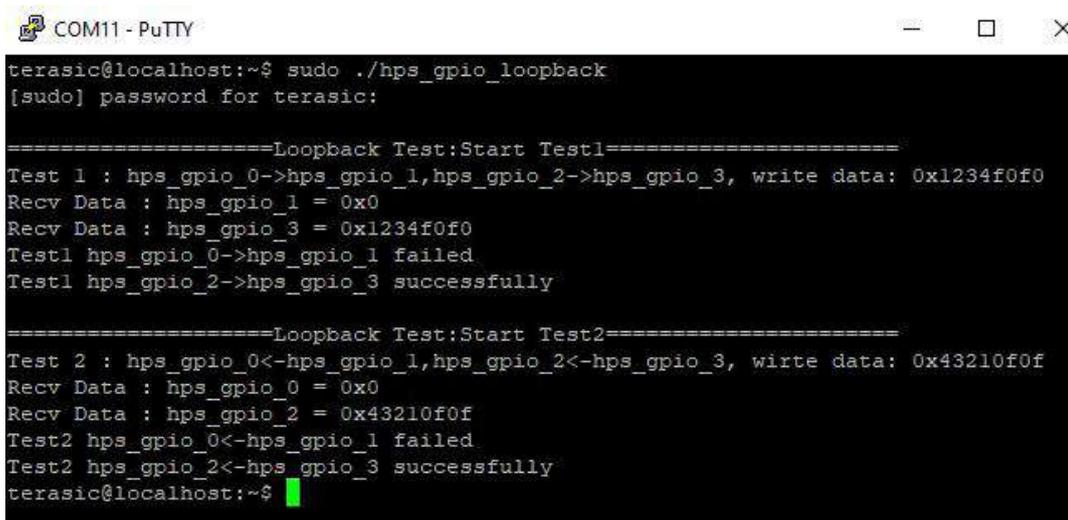
Figure 3-6 LED/KEY test

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.

Apollo S10 SoM
Demonstration
Manual

35

www.terasic.com
October 14,
2020
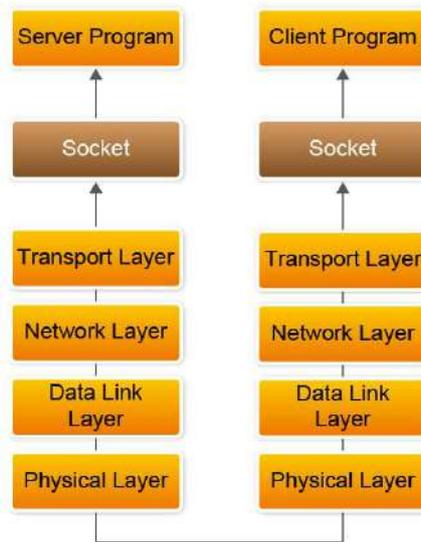
Figure 3-7 Communicate on a network via a socket

◼ 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.
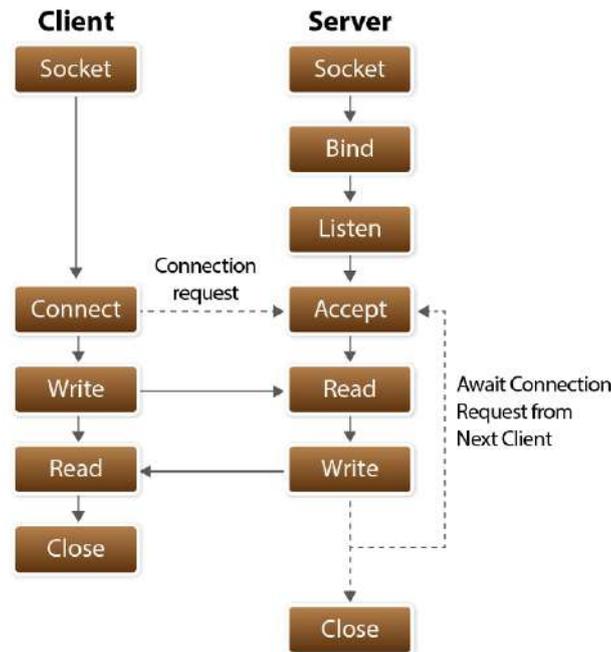
Figure 3-8 Client and Server communication

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

37

terasic
www.terasic.com

Apollo S10 SoM
Demonstration
Manual

www.terasic.com
October 14,
2020

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 3-9** 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 IP4v 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.

Apollo S10 SoM
Demonstration
Manual

39

www.terasic.com
October 14,
2020
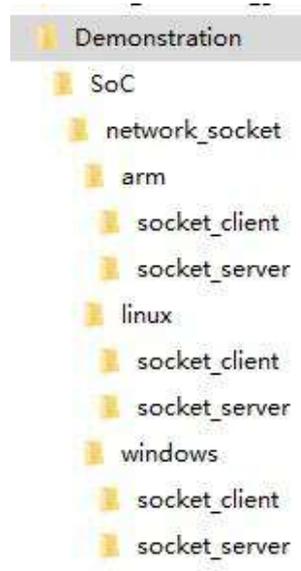
Figure 3-11 Source Code Folder Tree

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 S10 Module to Network via Ethernet port (J3).
2. Connect a USB cable to the Mini USB connector (J8) on the Apollo S10 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 S10 Module Linux BSP has pre-installed this code, so users can skip this copy action.)
4. Insert the Apollo S10 Module Linux BSP micro SD card into the Apollo S10 Module.
5. Power on the Apollo S10 Module.
6. In Windows, launch the Putty to connect Apollo S10 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**.
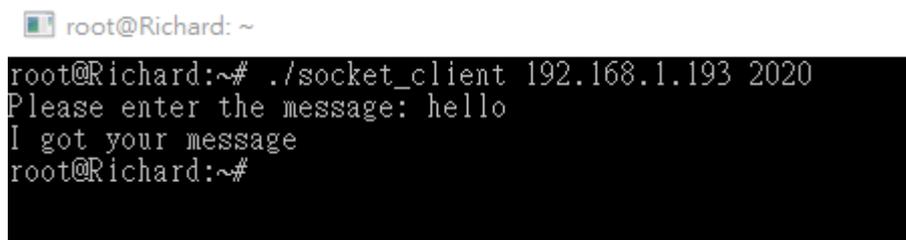
Apollo S10 SoM
Demonstration
Manual

41

www.terasic.com
October 14,
2020

Figure 3-14 Send Message in Client Program

terasic@linux:~$ ./socket_sever 2020
Here is the message: hello

terasic@linux:~$

Figure 3-15 Server dumps received message

3.4 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 3-16 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.
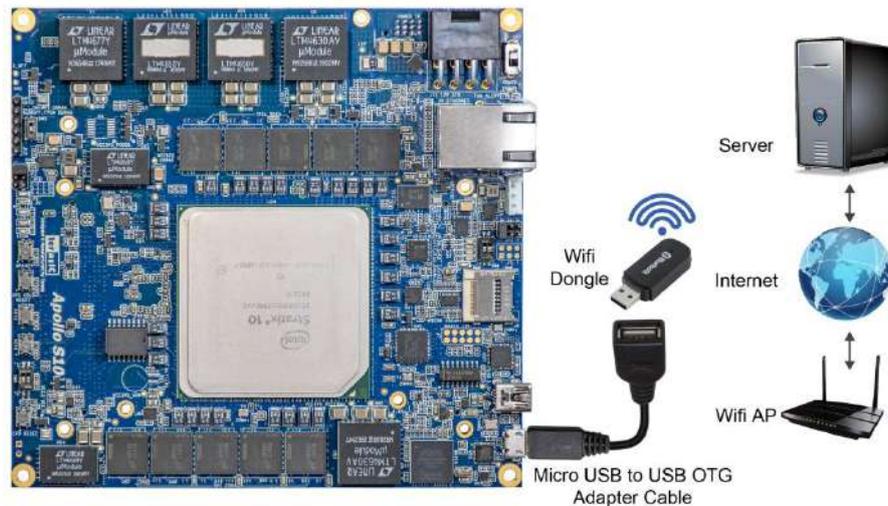
Figure 3-16 System diagram of USB Wi-Fi dongle

■ Wi-Fi Setup Procedure

1. Connect a USB cable to the Micro USB connector (J9) on the Apollo S10 and the Host PC.
2. Connect the USB Wi-Fi Dongle into the Micro USB connector (J9) on the Apollo S10 with USB Transfer Cable.
3. Power on the Apollo S10.
4. Launch Putty to establish the connection between the UART port of Apollo S10 and the Host PC.
5. In the Putty UART terminal, type user name "terasic" and password "123" to login Linux.
6. Type "**sudo ifconfig wlan0 up**" in the UART terminal of Putty to start wlan0 network interface. Input password "123" if system query password for terasic.
7. Type "**sudo 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 3-17**.

Apollo S10 SoM
Demonstration
Manual

43

www.terasic.com

October 14,
2020

Figure 3-17 Wi-Fi AP information

8. Type "**sudo vim /etc/wpa_supplicant/wpa_supplicant.conf**" in the UART terminal to edit Wi-Fi configuration file, as shown in **Figure 3-18.**

Figure 3-18 Edit Wi-Fi configuration File

9. 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 3-19**.

Figure 3-19 Replace ssid and psk

10. Type "**sudo ifup wlan0**" in the UART terminal to connect to the Wi-Fi AP, as shown in **Figure 3-20**.

11. Type "**ifconfig wlan0**" in the UART terminal to confirm an IP Address is assigned

Apollo S10 SoM
Demonstration
Manual

44

www.terasic.com
October 14,
2020

to wlan0 interface, as shown in **Figure 3-21**.

12. 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 3-22**.

```
terasic@localhost:~$ sudo ifup wlan0
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlan0/f0:b4:29:3c:eb:7a
Sending on   LPF/wlan0/f0:b4:29:3c:eb:7a
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 3 (xid=0xab8e5542)
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 7 (xid=0xab8e5542)
DHCPREQUEST of 192.168.1.110 on wlan0 to 255.255.255.255 port 67 (xid=0x42558eab
)
DHCPOFFER of 192.168.1.110 from 192.168.1.1
DHCPACK of 192.168.1.110 from 192.168.1.1
cmp: EOF on /tmp/tmp.JmknJr1ud8 which is empty
bound to 192.168.1.110 -- renewal in 1374 seconds.
terasic@localhost:~$
```

Figure 3-20 Type " sudo ifup wlan0"

```
terasic@localhost:~$ ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.110  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::f2b4:29ff:fe3c:eb7a  prefixlen 64  scopeid 0x20<link>
        ether f0:b4:29:3c:eb:7a  txqueuelen 1000  (Ethernet)
        RX packets 1951  bytes 499332 (499.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 6187 (6.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

terasic@localhost:~$
```

Figure 3-21 Type "ifconfig wlan0"

```
terasic@localhost:~$ 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 ti
me=159 ms
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=2 ttl=51 ti
me=151 ms
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=3 ttl=51 ti
me=152 ms
64 bytes from li92-186.members.linode.com (74.207.250.186): icmp_seq=4 ttl=51 ti
me=156 ms

--- www.terasic.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 151.558/154.885/159.116/3.219 ms
terasic@localhost:~$
```

Figure 3-22 Type "ping -c 4 www.terasic.com"

Apollo S10 SoM
Demonstration
Manual

45

www.terasic.com

October 14,
2020

3.5 HPS Control FPGA LED

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. This demonstration can be found in the path: System CD\Demonstrations\SoC_FPGA\hps_fpga_led\

■ 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 3-23** shows led_pio's address in Address Map. You can define a macro for the address when you use it.

Figure 3-23 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 3-24** 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: <Path to SoC EDS installation>\embedded\ip\altera\hps\ armv8\hwlib\include\soc_s10\socal.

The second offset address is 0x1000+0x80 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**.

Apollo S10 SoM
Demonstration
Manual

www.terasic.com

October 14,
2020

```
#define HW_REGS_BASE ( ALT_FPGA_BRIDGE_LWH2F_OFST ) // 0xf9000000
#define HW_REGS_SPAN ( 0x200000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

#define LED_PIO_BASE (0x1000+0x80)
#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)(HW_REGS_MASK));
```

Figure 3-24 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 3-25** 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 Apollo S10 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.

terasic
www.terasic.com

Apollo S10 SoM
Demonstration
Manual

48

www.terasic.com
October 14,
2020

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 3-25 LED PIO memory map code

■ **Main Program**

In the main program, the LED is controlled to perform LED light shifting operation as shown in **Figure 3-26**. When finishing 60 times of shift cycle, the program will be terminated.

```
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 3-26 C Program for LED Shift Operation

■ **Makefile and compile**

Figure 3-27 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_s10
SOCEDS_ROOT ?= $(SOCEDS_DEST_ROOT)
HWLIBS_ROOT = $(SOCEDS_ROOT)/ip/altera/hps/armv8/hwlib
CROSS_COMPILE = ~/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
CFLAGS = -g -Wall    -D$(ALT_DEVICE_FAMILY) -I$(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY)    -I$(HWLIBS_ROOT)/include/
LDFLAGS =  -g -Wall
CC = $(CROSS_COMPILE)gcc

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

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

Figure 3-27 Makefile content

To compile the project, type "make" in the command shell as shown in **Figure 3-28**. Then, type "ls" to check the generated ARM execution file "hps_fpga_led".

```
user@DESKTOP-RMUP9R6:/mnt/d/Demonstration/SoC_FPGA/hps_fpga_led$ make
~/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc -g -Wall
 -Dsoc_s10 -I/mnt/i/intelFPGA_pro/19.1/embedded/ip/altera/hps/armv8/hwlib/include/soc_s1
0   -I/mnt/i/intelFPGA_pro/19.1/embedded/ip/altera/hps/armv8/hwlib/include/ -c main.c -o
 main.o
~/gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc -g -Wall
  main.o -o hps_fpga_led
user@DESKTOP-RMUP9R6:/mnt/d/Demonstration/SoC_FPGA/hps_fpga_led$ ls
Makefile  hps_fpga_led  main.c  main.o
user@DESKTOP-RMUP9R6:/mnt/d/Demonstration/SoC_FPGA/hps_fpga_led$
```

Figure 3-28 ARM C Project Compilation

Apollo S10 SoM
Demonstration
Manual

50

www.terasic.com

October 14,
2020

■ **Execute the Demo**

To execute the demo, please boot the Linux from the SD-card in Apollo S10 board. Copy the execution file "hps_fpga_led" to the Linux directory, and type "chmod +x hps_fpga_led" to add execution attribute to the execute file. Then, type "./hps_fpga_led" to launch the ARM program. The LED[1:0] on Apollo S10 board 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: *Apollo_S10_SoM_Linux_Booting_Started_Guide.pdf*

3.6 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 "xz -d gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar.xz"
4. Type "tar xf gcc-linaro-7.2.1-2017.11-x86_64_aarch64-linux-gnu.tar"

Here is the procedure to compile the example project:

1. Login Linux or WSL on Windows.
2. Execute the .sh file in the path:

   <Path to SoC EDS installation>/embedded/embedded_command_shell.sh
3. Copy the project into the Linux System and go to the project folder.
4. Type "make" to build project as shown in **Figure 3-29.**

Figure 3-29 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**

9F., No.176, Sec.2, Gongdao 5th Rd,

East Dist, HsinChu City, Taiwan, 30070

Email: support@terasic.com

Web: www.terasic.com

Apollo S10 Web: S10.terasic.com

■ **Revision History**

Date	Version	Changes
2020.03	First publication	
2020.04	V1.1	Modify review item
2020.06	V1.2	Add section 2.5
2020.10	V1.3	Add Section 3.5 HPS Control FPGA LED

terasic
www.terasic.com

Apollo S10 SoM
Demonstration
Manual

52

www.terasic.com
October 14,
2020