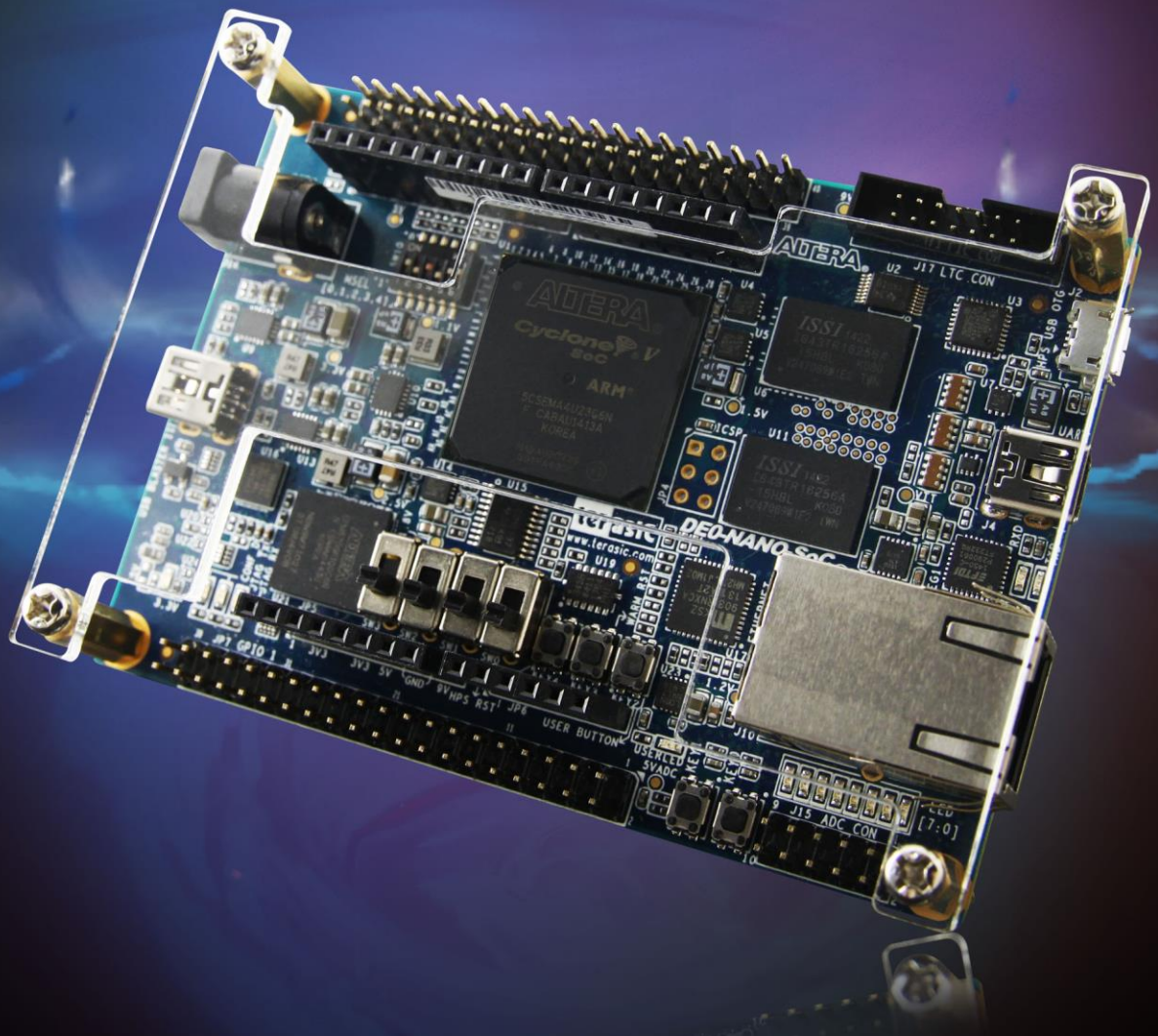
DE0-Nano-SoC

My First HPS-FPGA



terasIC
www.terasic.com

ALTERA
UNIVERSITY
PROGRAM

Copyright © 2003-2015 Terasic Inc. All Rights Reserved.

CONTENTS

Chapter 1

Overview

This tutorial introduces to SoC FPGA Beginners how to use the HPS/ARM to communicate with FPGA. The “My First HPS-FPGA” project demonstrates the implementation details. This project includes one Quartus project and one ARM C Project which demonstrates how HPS/ARM program controls the ten LEDs connected to FPGA.

Before reading this tutorial, developers need to read the documents below：

- DE0-Nano-SoC_Getting_Started_Guide.pdf
- My_First_Fpga.pdf
- My_First_HPS.pdf

For tutorial purpose, this document asks developers to create a HPS enabled Quartus project based on the project named **my_first_hps-fpga_base**. However, for the development of a formal HPS enabled project, developers are expected to create a Quartus project based on the **DE0-Nano-SoC-GHRD** (Golden Hardware Reference Design) project, which is included in the SYSTEM CD.

1.1. Required Background

This tutorial pre-assumed the developers have the following background knowledge:

■ FPGA RTL Design

- Basic Quartus II operation skill
- Basic RTL coding skill
- Basic Qsys operation skill
- Knowledge about Altera Memory-Mapped Interface

- **C Program Design**
  - Basic Altera SoC EDS(Embedded Design Suite) operation skill
  - Basic C coding and compiling skill
  - Skill to Create a Linux Boot SD-Card for DE0-Nano-SoC with a given image file
  - Skill to boot Linux from SD-Card on DE0-Nano-SoC
  - Skill to cope files into Linux file system on DE0-Nano-SoC
  - Basic Linux command operation skill

1.2. System Requirements

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

- **Altera DE0-Nano-SoC FPGA board, includes**
  - Mini USB Cable for UART terminal
  - Micros SD-Card, at 4GB
  - Micros SD-Card Card Reader
- **A x86 PC**
  - Windows 7 Installed
  - One USB Port
  - Quartus II 14.1 or Later Installed
  - Altera SoC EDS 14.1 or Later Installed
  - Win32 Disk Imager Installed

1.3. Altera SoC FPGA

In Altera SoC FPGA, the HPS logic and FPGA fabric are connected through the AXI (Advanced eXtensible Interface) bridge. For HPS logic to communicate with FPGA fabric, Altera system integration tool **Qsys** should be used for the system design to add Altera **HPS** component. From the AXI master port of the HPS component, HPS can access those Qsys components whose memory-mapped slave ports are connected to the master port.

The HPS contains the following HPS-FPGA AXI bridges:

- **FPGA-to-HPS Bridge**
  - HPS-to-FPGA Bridge
  - Lightweight HPS-to-FPGA Bridge

Figure 1-1 shows a block diagram of the AXI bridges in the context of the FPGA fabric and the L3 interconnect to the HPS. Each master (M) and slave (S) interface is shown with its data width(s). The clock domain for each interconnect is noted in parentheses.

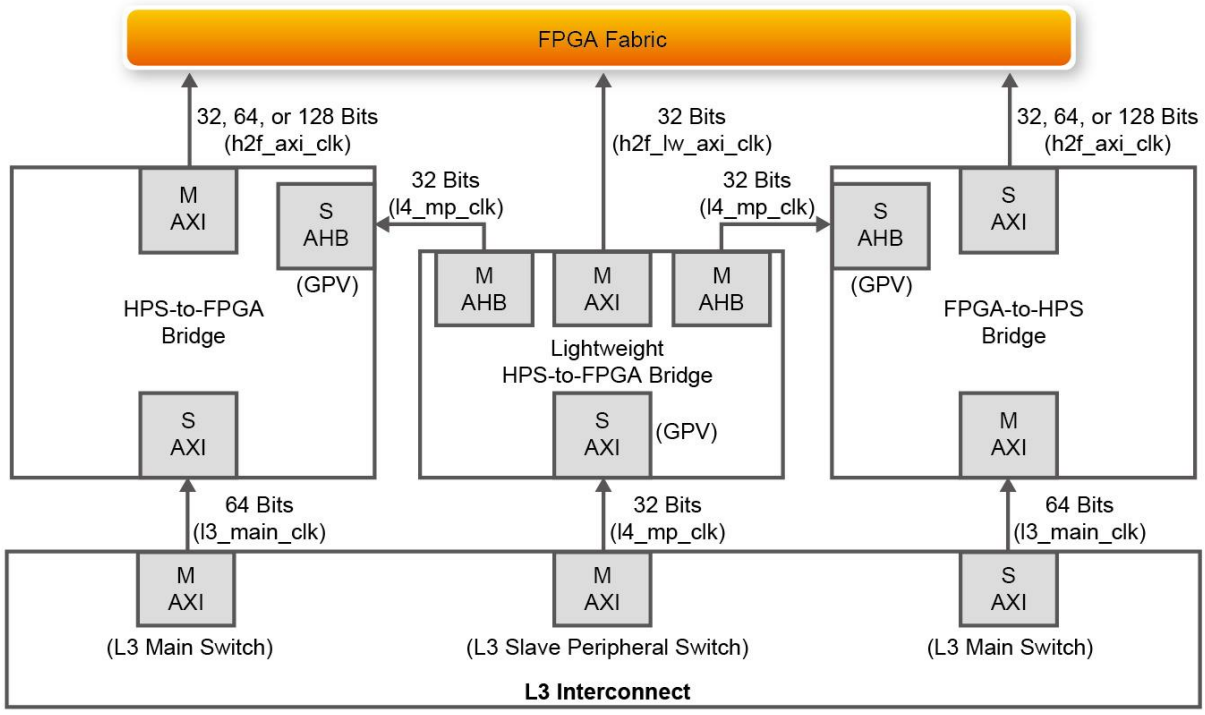


Figure 1-1 AXI Bridge Block Diagram

The HPS-to-FPGA bridge is mastered by the level 3 (L3) main switch and the lightweight HPS-to-FPGA bridge is mastered by the L3 slave peripheral switch. In the Quartus, HPS-to-FPGA bridge is used for ARM/HSP to control the LEDs which is connected to FPGA.

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

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

1.4. Source Code

The demonstration source codes include a Quartus project and a C project. They are located in the folder:

CD-ROM\Demonstration\SOC_FPGA\my_first_hps-fpga

The Quartus Project is located in the sub-folder “fpga-rtl” and the C project is located in the sub-folder “hps-c”. In this tutorial, developer are expected to establish these projects from a new one..

Chapter 2

Quartus Project

This chapter introduces how the **MY First HSP-FPGA** Quartus project is created based on the **my_first_hps-fpga_base** Quartus project. Base on this Quarturs project, a PIO component for controlling LED is added, a connection between the slave port of the PIO component and the master port of HPS component is established.

2.1. my_first_hps-fpga_base Quaruts Project

my_first_hps-fpga_base Quartus project is located in the DE0-Nano-SoC System CD folder:

CD-ROM\Demonstration\SOC_FPGA\my_first_hps-fpga_base

This Quartus project includes all required pin declearing for both HPS and FPGA. Note, the pin declaration of HPS needs to specify pin direction and IO standard. Pin location is not required for the pin declaration of HPS. The golden project also includes basic Qsys system which already includes a HPS component. The HPS component has been well-configured according to hardware design of DE0-Nano-SoC HPS.

Developers can open the Qsys system by opening the Quartus project, and clicking the menu item "Tools→Qsys" in Quartus II. When Qsys tool is launched, it will ask user to select a target Qsys system file. In this case, please select the Qsys file "soc_system.qsys". **Figure 2-1** shows the content of soc_system.qsys Qsys system. It contains hps_0 HPS component.

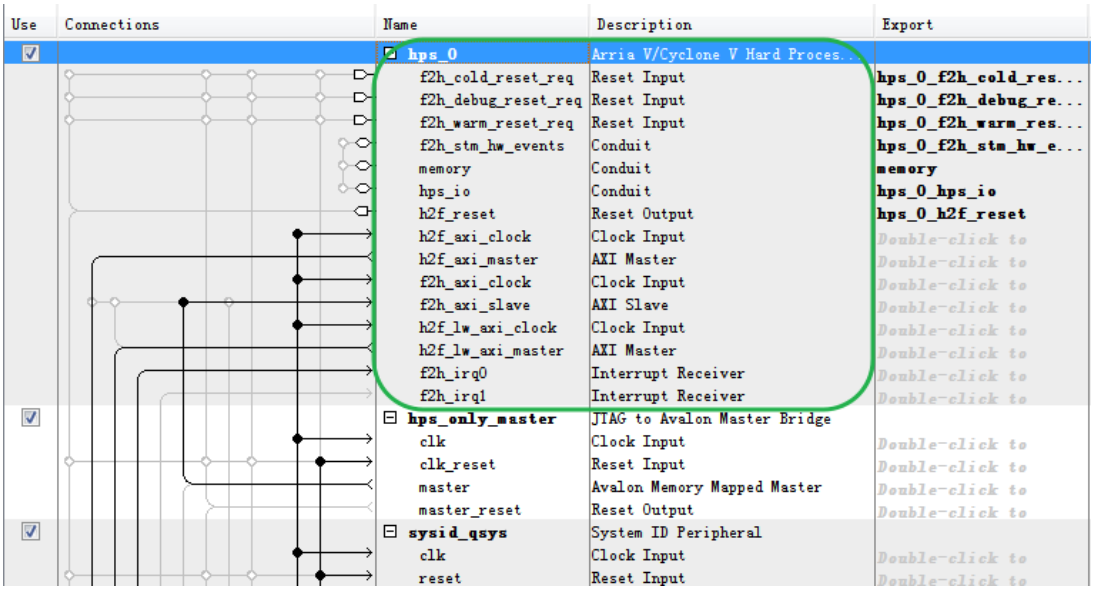


Figure 2-1 hps_0 HSP Component in Qsys System

Figure 2-2 shows the lightweight HPS-to-FPGA AXI Master port of the HPS component. Developers can connect this port to any memory-mapped slave port of components which developer wish to access from HPS/ARM.

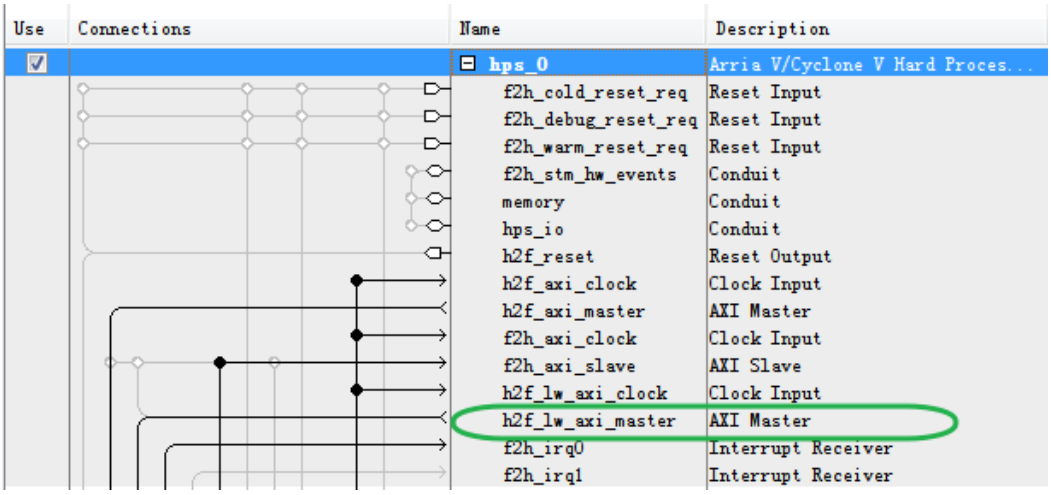


Figure 2-2 AXI Master Port of HPS component

2.2. Create a Quartus Project

This section will show how to add a PIO component in Qsys and how to connect the PIO component to the HPS component. The PIO component is used to control the ten red LEDs connected to FPGA. First, please copy the **my_first_hps-fpga_base** Quartus project to local disk. Open the project and open the Qsys system file "soc_system.qsys".

In the Library dialog of Qsys tool, enter 'pio' search key as shown in **Figure 2-3**. When "PIO (Parallel I/O)" appears, select it. Then, click "Add…" to add the PIO component to the system.

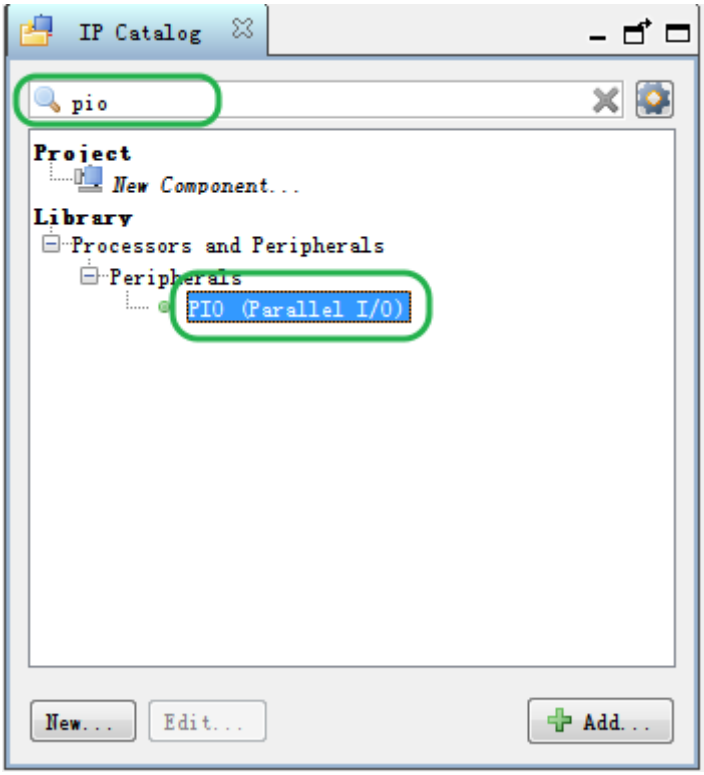


Figure 2-3 Find and Add PIO Component

When PIO dialog appears, please change **Width** to 8, make sure "Output" **Direction** is selected, and change the **Output Port Reset Value** to 0xff as shown in **Figure 2-4.**

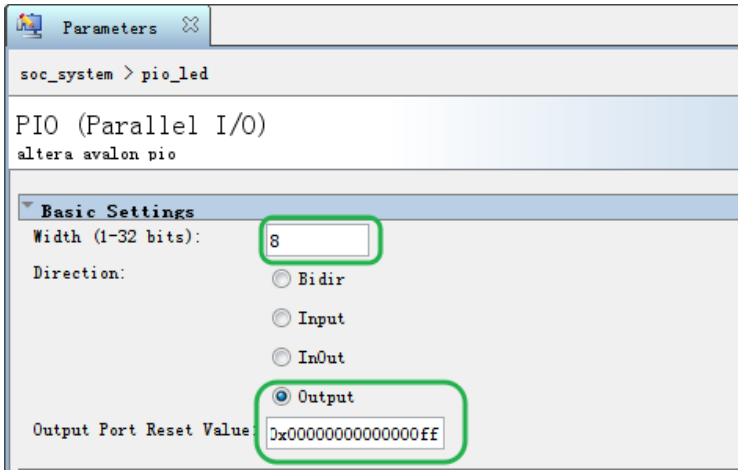


Figure 2-4 Configure PIO Component

When the PIO component is added into the system, please connect the h2f_lw_axi_master AXI master port to the s1 slave port of the PIO component as shown in Figure 2-5. By the way, please change PIO component name to **pio_led**, change the **Clock Input** to **clk_0**, export the **Conduit** signal as **pio_led_external_connection**, and connect the **Reset Input** to system reset. Note, the **Base** address of pio_led PIO component is very important. The ARM program will access the component according to this base address. In this demonstration, the base address is fixed at 0x0000_0000. The ARM program developer should remember this base address or use a given Linux shell batch file to extract the address information to a header file hps_0.h. The detail procedure will be described in the next chapter.

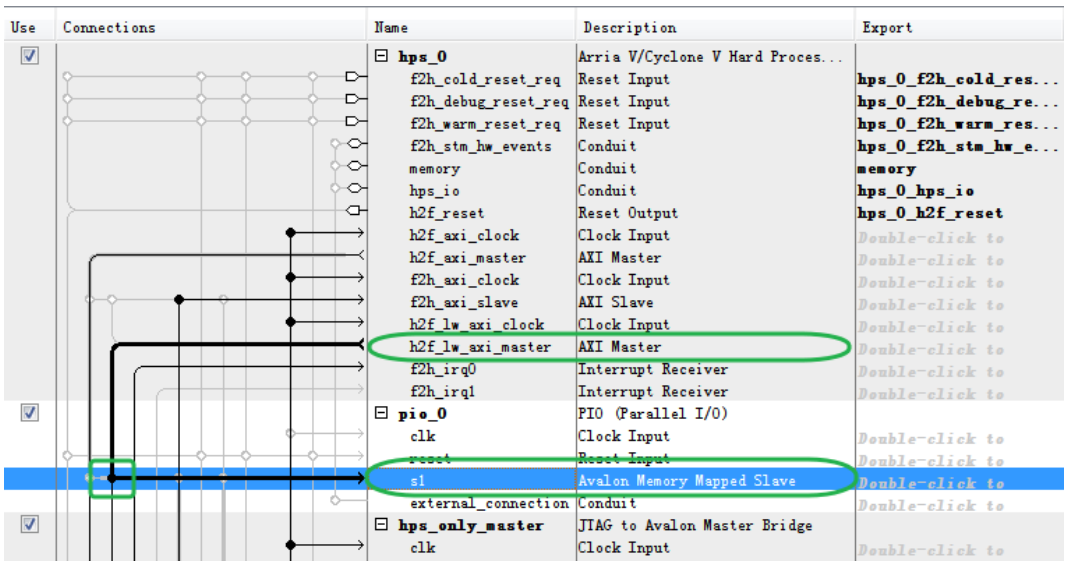


Figure 2-5 Create Connection Between HPS and PIO component

In the Qsys tool, click menu item "Generate→HDL Example…" can find the new interface signal **pio_led_external_connection_export** for the added **pio_led** PIO component as shown in **Figure 2-6**. Developer can click 'Copy' to copy the content to a clipboard, then paste the **pio_led_external_connection_export** signal to Quartus top and connect it to the **LEDR** port as shown in **Figure 2-7**. Before closing the Qsys tool, please remember to click the menu item "Generate→Generate…" to generate source code for the system.

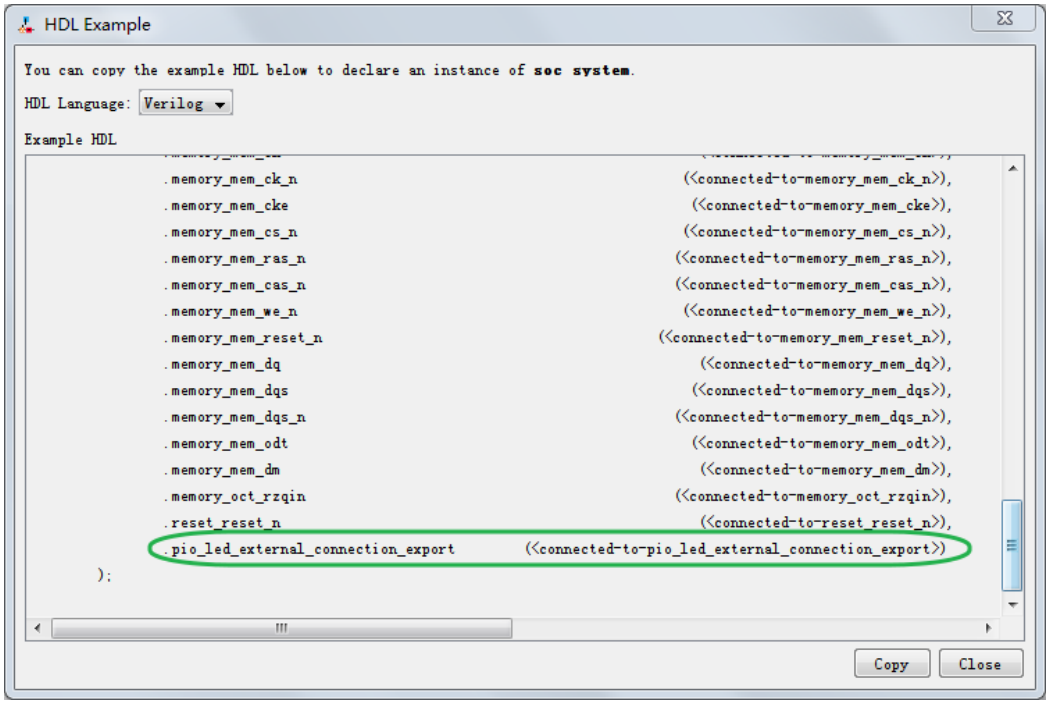


Figure 2-6 pio_led Interface of System

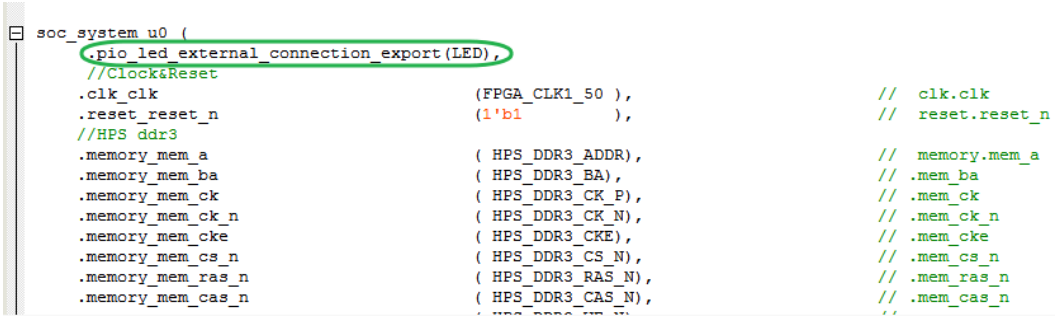


Figure 2-7 Initialize .pio_led_external_commection_export in u0 soc_system

2.3. Compile and Programming

Now, developers can start the compile process by clicking the menu item "Processing→Start Compilation". When the compilation process is completed successfully, **soc_system.sof** is generated. Developers can use this file to configure FPGA by Quartus Programming through the DE0-Nano-SoC on-board USB-Blaster II.

Because .tcl files of SDRAM DDR3 controller for HPS had been executed in my_frist_hps-fpga_base Quartus project, so developers can skip these projects. If developers' Quartus project is not developed based on the my_frist_hps-fpga_base Quartus project, please remember to execute the .tcl files, as show in **Figure 2-8**, before executing 'Start Compilation'. The TCL Scripts dialog can be launched by clicking the menu item "Tools→TCL Scripts…". <qsys_system_name>**_parameters.tcl** and <qsys_system_name>**_pin_assignments.tcl** tcl files should be executed, where <qsys_system_name> is the name of your Qsys system. Run this script to assign constrains tor the SDRAM DDR3 component.

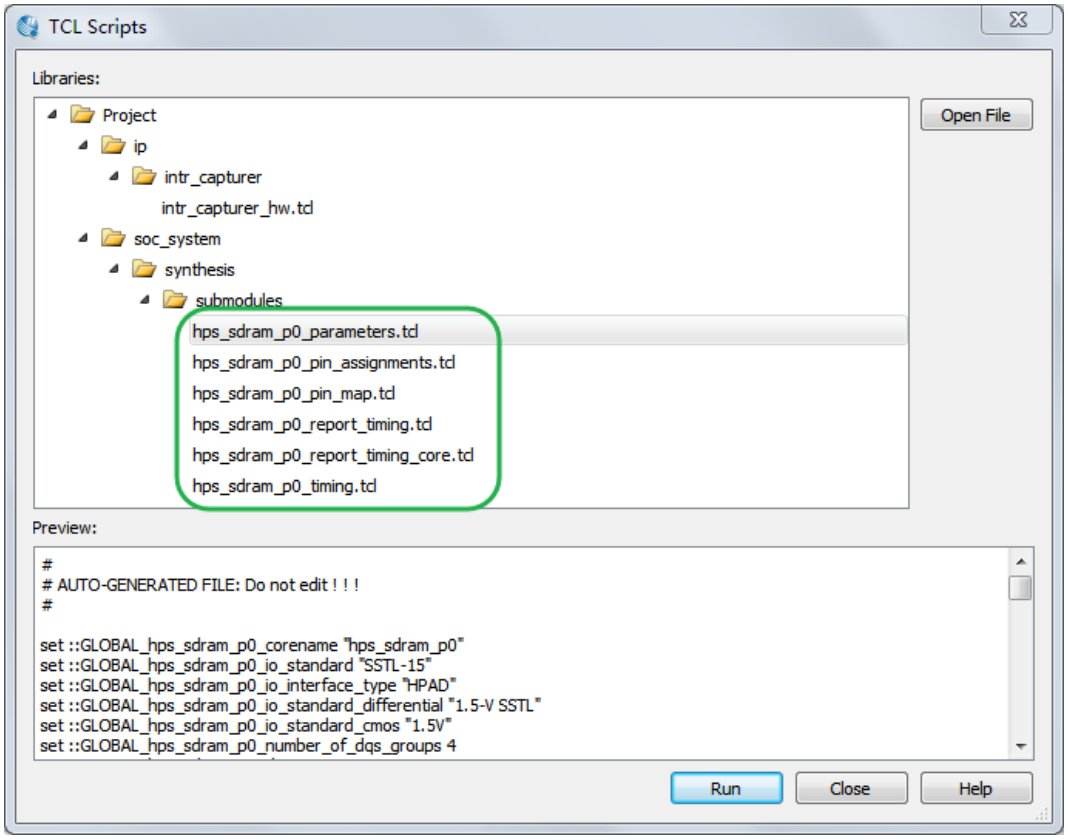


Figure 2-8 TCL file for SDRAM DDR3 of HPS

Chapter 3

C Project

This chapter introduces how to design an ARM C program to control the **pio_led** PIO controller. Altera SoC EDS is used to compile the C project. For ARM program to control the **pio_led** PIO component, **pio_led** address is required. The Linux built-in driver '/dev/mem' and **mmap** system-call are used to map the physical base address of **pie_led** component to a virtual address which can be directly accessed by Linux application software.

3.1. HPS Header File

pio_led component information is required for ARM C program as the program will attempt to control the component. This section describes how to use a given Linux shell batch file to extract the Qsys HPS information to a header file which will be included in the C program later.

The batch file mentioned above is called as **generate_hps_qsys_header.sh**. It is located in the same folder as my_first_hps-fpga Quartus project. To generate the header file, launch Altera SoC EDS command shell, go to the Quartus project folder, and execute **generate_hps_qsys_header.sh** by typing './generate_hps_qys_header.sh". Then, press ENTER key,a header file **hps_0.h** will be generated. In the header file, the **pio_led** base address is represented by a constant PIO_LED_BASE as show in **Figure 3-1**. The **pio_led** width is represented by a constant PIO_LED_DATA_WIDTH. These two constants will be used in the C program demonstration code.

```
/*
 * Macros for device 'pio_led', class 'altera_avalon_pio'
 * The macros are prefixed with 'PIO_LED_'.
 * The prefix is the slave descriptor.
 */
#define PIO_LED_COMPONENT_TYPE altera_avalon_pio
#define PIO_LED_COMPONENT_NAME pio_led
#define PIO_LED_BASE 0x0
#define PIO_LED_SPAN 32
#define PIO_LED_END 0x1f
#define PIO_LED_BIT_CLEARING_EDGE_REGISTER 0
#define PIO_LED_BIT_MODIFYING_OUTPUT_REGISTER 0
#define PIO_LED_CAPTURE 0
#define PIO_LED_DATA_WIDTH 8
#define PIO_LED_DO_TEST_BENCH_WIRING 0
#define PIO_LED_DRIVEN_SIM_VALUE 0
#define PIO_LED_EDGE_TYPE NONE
#define PIO_LED_FREQ 50000000
#define PIO_LED_HAS_IN 0
#define PIO_LED_HAS_OUT 1
#define PIO_LED_HAS_TRI 0
#define PIO_LED_IRQ_TYPE NONE
#define PIO_LED_RESET_VALUE 255
```

Figure 3-1 pio_led information defined in hps_0.h

3.2. Map pio_led Address

This section will describe how to map the pio_led physical address into a virtual address which is accessible by an application software. Figure 3-2 shows the C program to derive the virtual address of **pio_led** 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**. Then, the virtual address of **pio_led** 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_LWFPGASLVS_OFST in the header hps.h. The hps.h is a header of Altera SoC EDS. It is located in the folder:

Quartus Installed Folder\embedded\ip\altera\hps\altera_hps\hwlib\include\socal

The second offset address is 0x00000000 which is defined as PIO_LED_BASE in the hps_0.h header file which is generated in above section.

The virtual address of pio_led 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 **pio_led**.

```
if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
  printf( "ERROR: could not open \"/dev/mem\"...\n" );
  return( 1 );
}

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" );
  close( fd );
  return( 1 );
}

h2p_lw_led_addr=virtual_base +
                ( ( unsigned long  )( ALT_LWFPGASLVS_OFST + PIO_LED_BASE )
                & ( unsigned long)( HW_REGS_MASK ) );
```

Figure 3-2 pio_led information defined in hps_0.h

3.3. LED Control

C programmers need to understand the Register Map of the PIO core for **pio_led** before they can control it. **Figure 3-3** 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 DE0-Nano-SoC is high active, writing a value 0x00000000 to the offset 0 register will turn off all of the eight green LEDs. Writing a value 0x000000ff to the offset 0 register will turn on all of eight green LEDs. In C program, writing a value 0x000000ff to the offset 0 register of pio_led can be implemented as:

*(uint32_t *) h2p_lw_led_addr= 0x000000ff;

The state will assign the void pointer to a uint32_t pointer, so C compiler knows write a 32-bit value 0x000000ff to the virtual address h2p_lw_led_addr.

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture *(1), (2)*		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

Figure 3-3 Register Map of PIO Core

3.4. Main Program

In the main program, the LED is controlled to perform LED light sifting operation as shown in Figure 3-4 . 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,  add ~ because the led is low-active
  *(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 << (PIO_LED_DATA_WIDTH-1)))
       led_direction = 1;
  }else{
    led_mask >>= 1;
    if (led_mask == 0x01){
      led_direction = 0;
      loop_count++;
    }
  }

} // while
```

Figure 3-4 C Program for LED Shift Operation

3.5. Makefile and compile

Figure 3-5 shows the content of Makefile for this C project. Because the program will include the hps.h provided by Altera SoC EDS, so the Makefile should include the following path:

"${SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib/include"

In the makefile, ARM cross-compile also be specified.

```
#
TARGET = my_first_hps-fpga

#
CROSS_COMPILE = arm-linux-gnueabihf-
CFLAGS = -static -g -Wall  -I${SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib/include
LDFLAGS =  -g -Wall
CC = $(CROSS_COMPILE)gcc
ARCH= arm


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

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

Figure 3-5 Makefile content

To compile the project, type "make" as shown in Figure 3-6. Then, type "ls" to check the generated ARM execution file "my_first_hps-fpga".

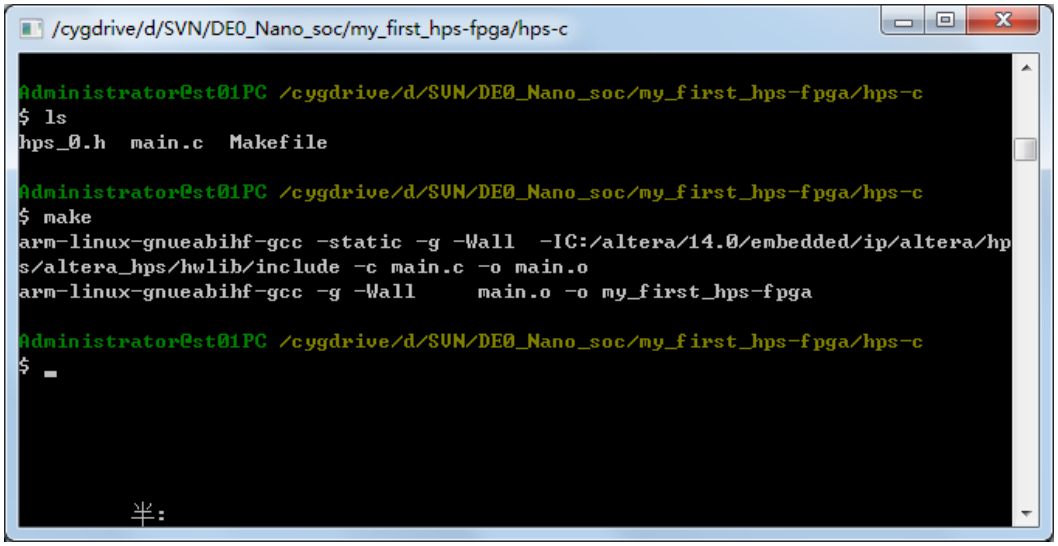


Figure 3-6 ARM C Project Compilation

3.6. Execute the Demo

To execute the demo, please boot the Linux from the SD-card in DE0-Nano-SoC. Copy the execution file “my_first_hps-fpga” to the Linux directory, and type “chmod +x my_first_hps-fpga” to add execution attribute to the execute file. Use Quartus Programmer to configure FPGA with the **soc_system.sof** generated in previous chapter. Then, type “./my_first_hps-fpga” to launch the ARM program. The LED on DE0-Nano-SoC 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:

DE0-Nano-SoC _Getting_Started_Guide.pdf

For details about copying files to Linux directory, please refer to the document:

My_First_HPS.pdf