LAB 2: And Array implemented in Programmable Logic

Introduction

This lab will introduce the use of the programmable logic available within the XC7020 chip. The logic will implement a two input AND gate, each input will be 8 bits wide. The processor will provide the data to the AND gate via a 16-bit wide GPIO. The 8 outputs of the AND gate will be displayed on the LEDs. Figure 1 gives the block diagram of the system.

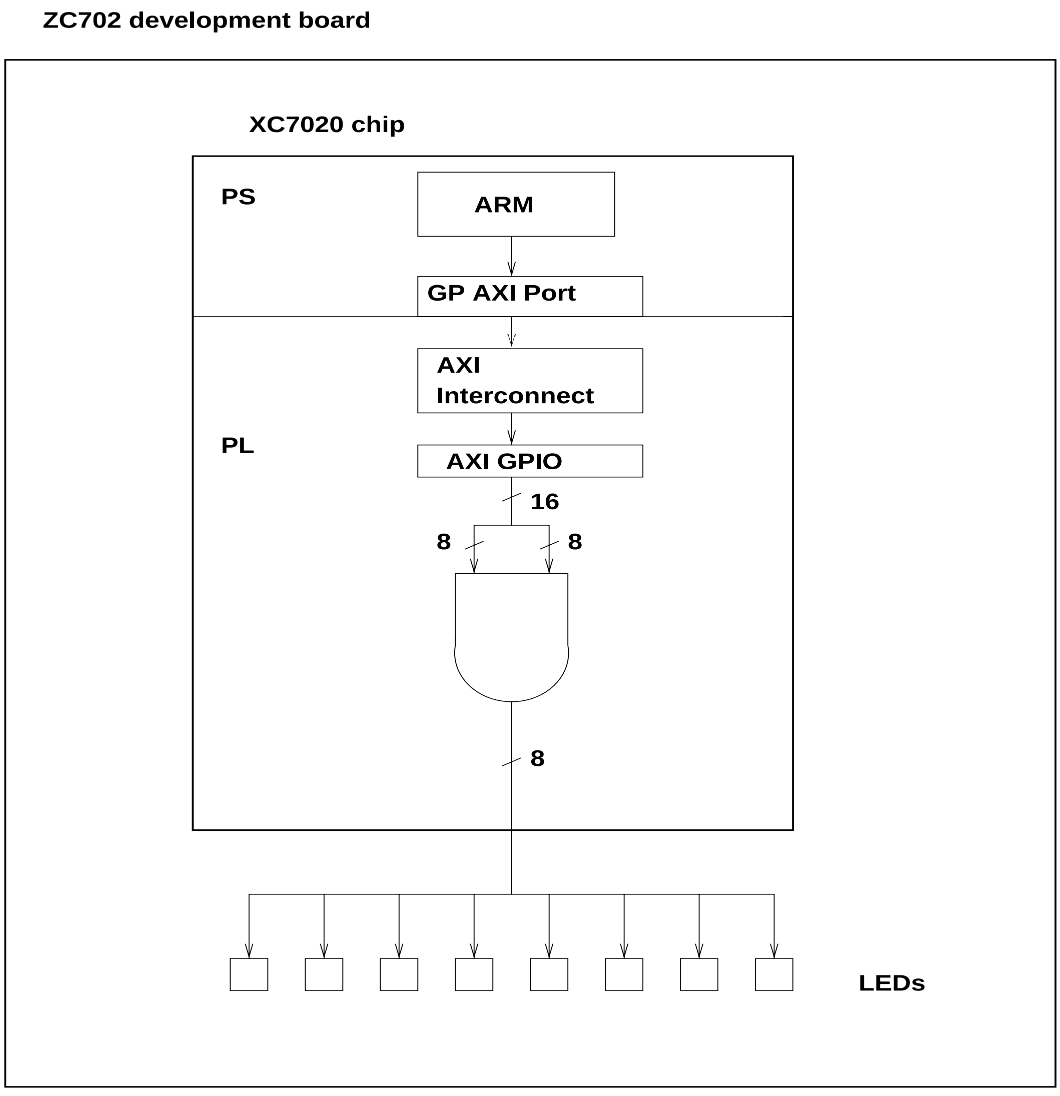

Figure 1: System block diagram.

Setting up the user Linux environment:

1. Type the following on a terminal:

source /CMC/tools/xilinx_14.7/14.7/ISE_DS/settings64_CMC_central_license.csh

This sets up the environment to run the tools.

2. Launch the planAhead tool:

planAhead &

A message may pop up:

A disk write failure occurred. There may be insufficient disk space or you may not have write permission at the following directory.
/nfs/sw_cmc/linux-64/tools/xilinx_14.4/14.4/ISE_DS/.xinstall
Press Retry to try again, press Cancel to exit XilinxNotify.

Press Cancel for now. The tool may be trying to write to the install directory.

Create a new project from PlanAhead:

3.1. File -> New Project

3.2. A window pops up, click Next.
Give your project a name and choose the desired project path.
Project name: lab2

Leave the "Create project subdirectory" selected.

Select Next.

3.3 RTL Project should be selected. Keep the selection.

Select Next

3.4. Add the and_gate8 source

For this lab, the VHDL code which implements the AND gate is provided in:

/home/t/ted/PUBLIC/COEN317/Lab2/lab2_AND.vhd

You may copy this file into your COEN317/LAB2 directory and add this file as the source file. Alternatively, you may use a text editor to create the file containing the following VHDL code:

```
-- 8-bit AND gate
library ieee;
use ieee.std_logic_1164.all;
```

```
entity and_gate8 is
port(   x: in std_logic_vector(0 to 7);
            y: in std_logic_vector(0 to 7);
            F: out std_logic_vector(0 to 7)
);
end and_gate8;

architecture Behavioral of and_gate8 is
begin
        F <= x and y;

end Behavioral;
```

Select Add Files -> lab2_AND.vhd
Target language: VHDL
Select Next

3.5. We do not need to specify any IPs

Select Next

3.6 Specify the UCF file for 8 LEDs. For this lab, the UCF file to be used is found in:

/home/t/ted/PUBLIC/COEN317/Lab2/lab2_gpio_for_C.ucf

Copy this file into your COEN317/Lab2 directory and specify it as the UCF file.

Select Add Files -> lab2_gpio_for_C.ucf
Select Next

3.7 Select the ZYNQ-7 ZC702 Evaluation Board for our project

Select Boards.
Scroll down the list and choose ZYNQ-7 ZC702 Evaluation Board.
Select Next.

.3.8 Review settings and click Finish

Create an embedded processor project with the Add Source wizard

4.1 On the left panel under Project Manager, select Add Sources.

4.2 Choose Add or Create Embedded Sources and Select Next.

4.3 Select Create Sub-design.

A window should have popped up asking for a module name. Name it “system”
Module name: system
Select OK.
Select Finish.

Design the system in XPS:

XPS is launched to set up the newly created system.

5.1 A window pops up and asks:
This project appears to be a blank zynq project.
Do you want to create a Base System using the BSB Wizard?

Select Yes.

5.2 The BSB Wizard asks for additional settings. The AXI System should be selected by default with no other parameters.

Select OK.

Verify the Zynq Processing System 7 is selected in “Select a System”
Select Next.

Remove the GPIO_SW and LEDs_4Bits Peripherals because they are not needed.
Click on “Select All”, then “< Remove”
Select Finish.

5.3 Add the AXI General Purpose IO for the AND gate inputs A and B.

Expand the General Purpose IO tab on the left hand side under IP Catalog
Double-click AXI General Purpose IO.

Click Yes for:
Do you want to add one axi_gpio 1.01.b IP instance to your design?

Rename Component Instance Name to axi_gpio_for_A_and_B.
Expand Channel 1 and lower the GPIO Data Channel Width to 16.

Select OK.

The selected processor instance to connect the GPIO to should be processing_system7_0.

Select OK.

5.4 Change the port settings for axi_gpio_for_A_and_B:

Select the **Ports** tab in the System Assembly View
Expand axi_gpio_for_A_and_B and its (IO_IF) gpio_0

Right-click GPIO_IO and select **No Connection**
Right-click GPIO_IO_O and select **Make External**

5.5 Close the XPS:

Select File -> Exit

Closing XPS moves the PlanAhead window to the foreground.

Exporting the Hardware to SDK:

You will notice in the top middle pane of Project Manager that there is a newly created Design Source: system (system.xmp) in addition to the (lab2_AND.vhd) file which was added with PlanAhead.

6.1 Right-click on "system (system.xmp)" and choose Create Top HDL

This will create the top-level VHDL file called system_stub.vhd found in :

./lab2/lab2.srcs/sources_1/edk/system/system_stub.vhd

This file consists of the top-level VHDL entity called system_stub together with a component specification for the system designed in XPS. The name of the component is system (the name chose in step 4.3). You will note that both the entity system_stub and the component system have a port of mode out called axi_gpio_for_A_and_B_GPIO_IO_O_pin. This is what step 5.4 performed (when we selected the port GPIO_IO_O and specified Make External. Had we desired, we could have renamed the port in the External Ports of XPS to same other name, since we did not choose to do so, the system left it as the default (and somewhat unwieldy) name axi_gpio_for_A_and_B_GPIO_IO_O_pin. The system_stub.vhd file will be modified to include our AND gate component.

6.2 Modify the created system_stub.vhd to include a component for the AND gate, and output signal, and an internal signal used to provide the two input values to the AND gate:

Add a top-level "output" as a output vector port to the system_stub entity (use 0 to 7 as the slice direction).
Add the and_gate8 component.
Add the signals signal_for_A_and_B and output_signal (use 0 to 7 for the slice).
Port Map signal_for_A_and_B to the system and and_gate8,
and_gate8 port maps F to output.

The file found in:

/home/t/ted/PUBLIC/COEN317/Lab2/lab2_stub.vhd

contains all the necessary modification. Copy this file into your

./lab2/lab2.srcs/sources_1/edk/system/system_stub.vhd

If PlanAhead generates a message stating that the system_stub.vhd file has changed, select Reload to load in the newly copied version.

6.3 Synthesize the VHDL code. Click on Run Synthesis on the left Panel.

6.4   Implement the Synthesis.
Choose Run Implementation and click OK.

There may be 3 constraint warnings, they do not affect our system and you can click OK. Wait for Implementation to finish. Figure 2 gives the RTL (register transfer level schematic) diagram obtained from the design implementation. To open the schematic from PlanAhead, select Schematic listed under RTL Analysis.

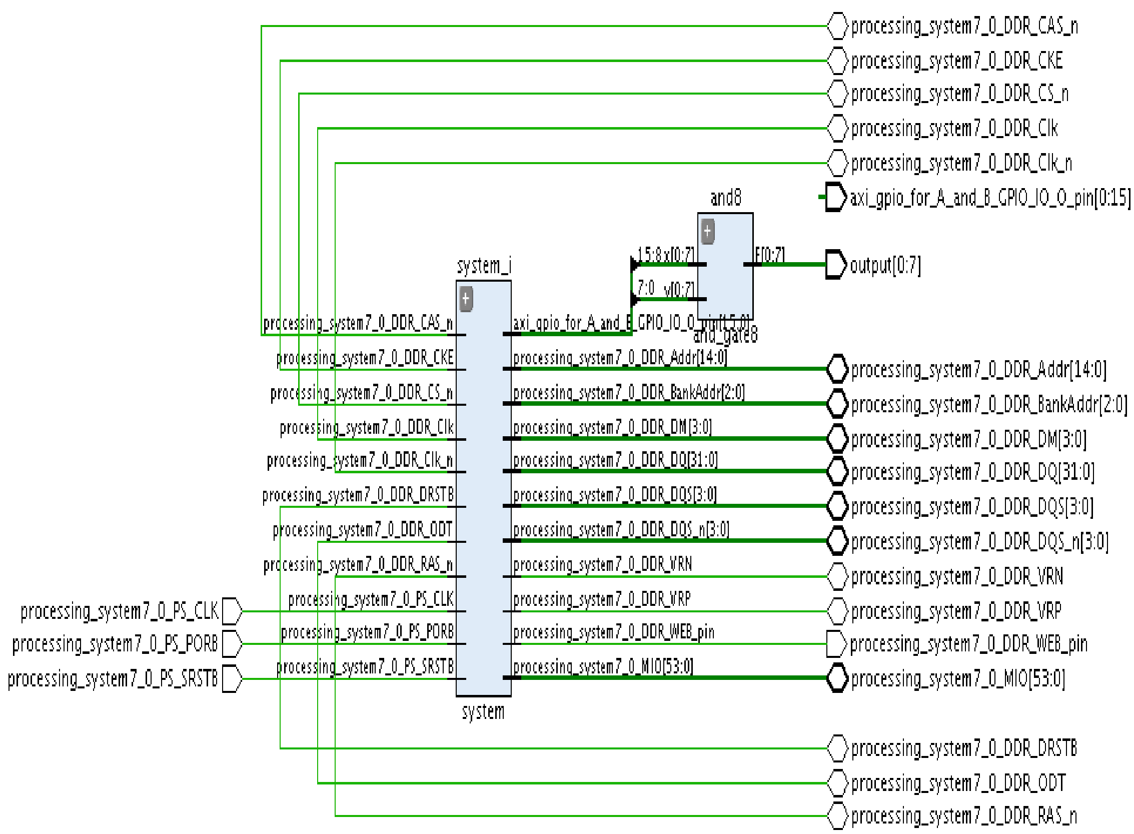

Figure 2: RTL Schematic of Implemented system.

6.5 Generate the bitstream. If necessary, specify the -g UncontrainedPins Allow option in the Bitstream Settings. Choose Generate Bitstream and click OK. Wait for bitstream generation to complete.

6.6 Download the bitstream to the Zynq Board. Attach the power cable, the Platform Cable USB II, and the serial cable for the UART. Apply power to the board and verify that the Platform Cable USB II status LED is illuminated in green.

Choose Launch iMPACT and click OK. A window may pop up asking you to save the file, click Cancel. In iMPACT, right-click the Target and choose Program. Ensure Device 2 (FPGA xc7z020) is highlighted and click OK.

When you see Program Succeeded, close iMPACT by selecting :

File -> Exit

You do not need save the iMPACT project: Click No

6.7 Go back to the PlanAhead tool:

File -> Export -> Export Hardware for SDK
Check "Launch SDK" and click OK on the pop up screen
SDK will open.
SDK may crash on the first the project is created.
(Check your terminal, you may see

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x0000000000000000, pid=7396, tid=47440467388736
```

)
If it crashed, repeat this step and overwrite the project

Using SDK to create an application project:

7.1 Create a new software project.

File -> New -> Application Project

Specify a project name and leave the default settings.
Project name: lab2
Click Next

Leave Empty Application project selected Template.
Click Finish

7.2 The C++ source code for this lab is found in :

/home/t/ted/PUBLIC/COEN317/Lab2/main.cc

Copy this file into your ./lab2/lab2.sdk/SDK/SDK_Export/lab2/src/main.cc.

The program sets up the GPIO port for output and initializes two boolean arrays. The eight values of each array are concatenated into two 8 bit numbers and then each of these two numbers is concatenated into a 16 bit number which is sent to the GPIO port.

7.3 Compile the program to build the executable.

7..4 Connect the SDK terminal to the board.

Window -> Show View -> Terminal

Click the green connect button (it reembles a green ‘N’ with a dot on each end)

On the popup, choose Serial as the Connection Type
The settings are:

Port: /dev/ttyUSB0
Baud Rate: 115200
Data Bits: 8
Stop Bits: 1
Parity: None
Flow Control: None
Timeout(sec): 5

Click OK.

Make sure it says “CONNECTED”

7.5 Run the executable file on the board.

Right-click the lab2 folder (NOT lab2_bsp) and choose Run As -> Run Configurations

On the popup window, right click Xilinx C/C++ ELF and choose New
(This only needs to be created once)

Leave the default settings and click Run
Go on the SDK Terminal to see the output. Observe the state of the 8 LEDs.

Questions:

1. In the main.cc program, change the the initial vlaues of the two boolean arrays:

```
bool A[8] = {0, 1, 0, 0, 1, 1, 1, 0};
bool B[8] = {1, 1, 1, 0, 0, 1, 1, 1};
```

to some other intial values. Recompile the code and download it to the board and observe the state of the 8 LEDs.

T. Obuchowicz/R.Lee, July 2014.

Revision History:

- Sept. 8, 2016: revised for Lab 2 Fall 2016 and sourcing /CMC/tools/xilinx_14.7/14.7/ISE_DS/settings64_CMC_central_license.csh