ECEN 449: Microprocessor System Design
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Sunil P Khatri
(Lab exercise created and tested by Ramu Endluri, He Zhou and Sunil P Khatri)

Laboratory Exercise #2

Using the Software Development Kit (SDK)

Objective

The purpose of lab this week is to familiarize you with Vivado by developing a software based solution for controlling the LEDs in a manner similar to that which was done last week in pure FPGA hardware. To accomplish this goal, you will be guided through the process of creating a MicroBlaze processor system using the Vivado Block Design Builder . You will then add General Purpose Input/Output (GPIO) capabilities to the microprocessor via Intellectual Property (IP) hardware blocks from Xilinx. Finally, you will create software using the C programming language, which will run on the MicroBlaze processor inorder to implement the appropriate LED functionality.

System Overview

The microprocessor system you will build in this lab is depicted in Figure 1. To the left of the diagram is the MicroBlaze soft IP processor. Connected to it are two Local Memory Buses (LMBs), iLMB and dLMB for instruction fetch and data access respectively. Each LMB has its own block RAM (BRAM) controller which provides the interconnect logic between the MicroBlaze and BRAM (local memory). The AXI interconnect connects the MicroBlaze (bus master) to peripherals (bus slaves) external to the microprocessor. Typically included in the list of peripherals are the debugger module. The debugger allows the SDK to interact with the MicroBlaze processor after the FPGA has been programmed. This is useful for initializing regions of

memory outside of the FPGA and for general software debugging. The GPIO blocks provide the microprocessor with a means of controlling the LEDs and reading user input from the DIP switches and push buttons.

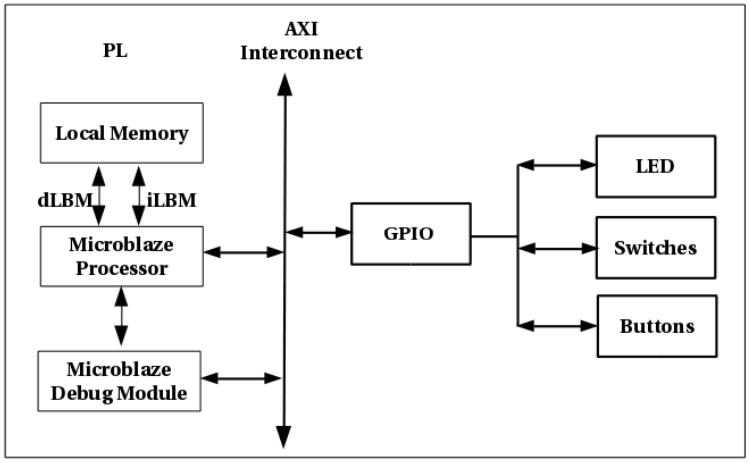

Figure 1: MicroBlaze System Diagram

Procedure

1. In the following steps we will launch Vivado and create a block design.

   (a) Before beginning, create a directory in your home folder for today's lab. Try to avoid spaces and special characters in the directory name as they have the potential for causing problems during the system build process. You may use the 'mkdir' command in an open terminal to create a directory.

   (b) Next, type the following commands in the terminal window:

   >source /softwares/Linux/xilinx/Vivado/2015.2/settings64.sh
   >vivado

   The first command will setup the environment for Vivado, and the second command runs Vivado.

   (c) Once Vivado launches, select 'Create New Project' and follow the same procedure as shown in Lab1 to create a new RTL project except do not add a source file in the project. In this lab you will use Xilinx Microblaze Processor.

(d) On the left side, in the Flow Navigator under the IP Integrator section, click on 'Create Block Design'. A window opens where you can specify the name of your design(eg. led_sw). Leave the 'Directory' and the 'Specify source set' as they are. Click on the 'OK' button. (Figure 2)

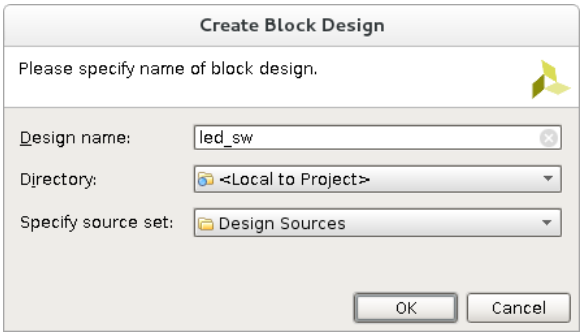

Figure 2: Create Block Design

(e) The Block Design Diagram opened (tab called Diagram) is empty. Right click within the diagram tab and select 'Add IP'. Search 'MicroBlaze' and double click on 'MicroBlaze' to add it to our design. Right click and select 'Run Block Automation'. Select the following configuration for the MicroBlaze processor and click 'OK'. (Figure 3)

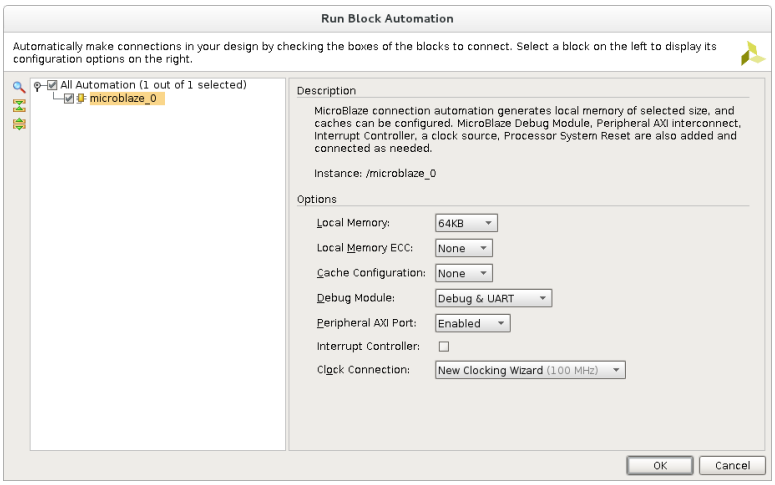

Figure 3: Run Block Automation

Local Memory: 64KB

Local Memory ECC: None
Cache Configuration : None
Debug Module: Debug & UART
Peripheral AXI Port: Enabled
Interrupt Controller: Unchecked.
Clock Connection: New Clocking Wizard (100 MHz)

(f) Once the Diagram is generated, double-click on the 'Clocking Wizard' block named 'clk_wiz_1'. This launches the 'Re-customize IP' window. Select 'Clocking Options' and change the source of the 'Primary Input Clock' from 'Differential clock capable pin' to 'Single ended clock capable pin'. Click 'OK'.

(g) Right click and select 'Run Connection Automation' in the diagram. In the 'Run Connection Automation window' check 'All Automation' and click 'OK'.

(h) The processor is set up, now we need to add General Purpose IO (GPIO) blocks to interact with LEDS, Switches and Buttons on the ZYBO board. Right click and select 'ADD IP'. Search for 'GPIO' and select 'AXI GPIO' to add it to the design. Double click on the GPIO block named 'axi_gpio_0' and select the 'IP configuration' tab. Set the following configuration for the GPIO and click 'OK' (Figure 4).

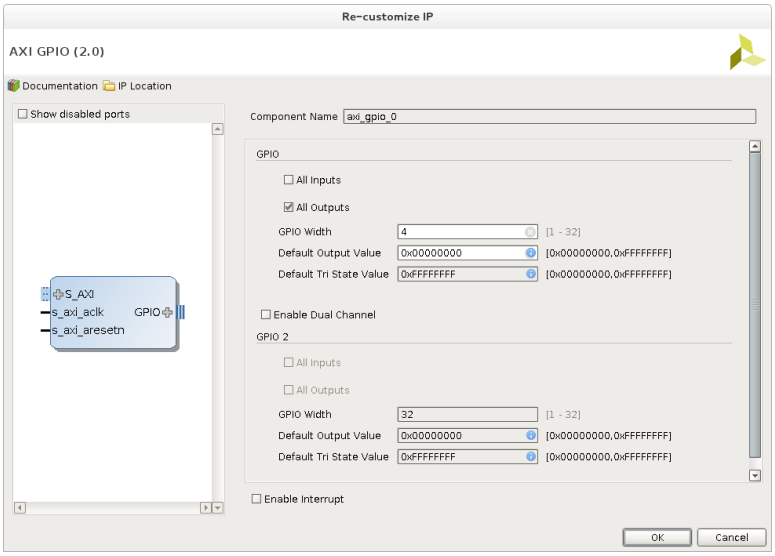

Figure 4: Customize GPIO

All Inputs: (not checked)

All Outputs (checked)
GPIO Width: 4
Leave the remaining values unchanged.

(i) Right click and select 'Run Connection Automation', check 'All Automation', and click 'OK'. Now design is connected. To make it look neat, right click and select 'Regenerate Layout'.

(j) You are using the GPIO block to configure LEDs, so let's rename the 'AXI GPIO' block. Select the 'AXI GPIO' block and rename the block to 'led' in the 'Block Properties' panel and press 'Enter'. Repeat the same procedure for the port connected to the 'AXI GPIO' block and rename it to 'led'. The layout should look like Figure 5

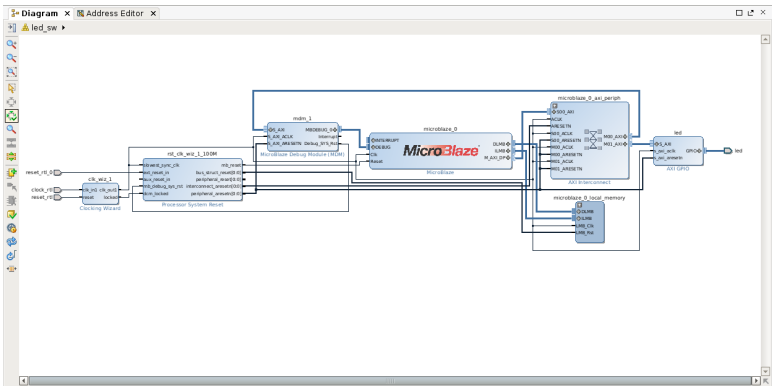

Figure 5: Layout

(k) Double click the 'reset_rtl_0' port in the diagram. You can see that it is 'ACTIVE LOW'. Double Click on the 'reset_rtl' port and observe that it is 'ACTIVE HIGH'.

(l) These are the reset ports for Microblaze. On ZYBO board we have dedicated reset pins for reset. Refer to manual and see the functionality of BTN6 and BTN7.

(m) These two reset ports can be configured to a push button on the board. For the purpose of this lab, connect the ports to constant values and use the dedicated pins on board for reset.

(n) Select 'Constant' IP from the IP repository (Right click and ADD IP) and add two instances to the diagram. Double click on one of the constant IP blocks and set 'Constant Width' to 1 and 'Constant Value' to 1. Rename this block to 'VDD'. Now, right click on the 'reset_rtl_0' port and select 'Delete' to delete it. Connect the constant block to 'ext_rst_in' of the 'Processor System Reset' because it is 'ACTIVE LOW'. Connect the constant block by clicking on the output of the constant block and dragging it to the 'ext_rst_in' pin.

(o) Repeat the same procedure for the other constant block but set 'constant value' to 0 and rename it as 'GND'. Delete the 'reset_rtl' port and connect it to the 'reset' port of the 'Clocking Wizard'.

(p) Right click on the diagram and select 'Regenerate Layout'. The final layout should look like Figure 6.

2. In the following steps we map the IO ports to the LEDs and buttons.

   (a) In the design tab, expand 'External Interfaces' and 'Ports' and examine the ports listed. When 'led' is expanded it shows the 'led_tri_o'and 'clock_rtl' ports listed. We need to connect these ports to the ZYBO board.

   (b) In the sources panel, right click on the constraints and select 'Add sources'. Next, in the add sources window, select 'Add or create constraints' and click 'Next'. Click on the green '+' button and select 'create file'. The 'Create Constraints File' window will open, give a file name (eg. led), and click 'OK'. Select 'Finish' to create a constraint file.

   (c) In the sources panel, expand 'Constraints' and double click on the created XDC file to open it.

   (d) Add the following code to attach the above ports to the pins on ZYBO board.

```
#clock_rtl
set_property PACKAGE_PIN L16 [get_ports clock_rtl]
set_property IOSTANDARD LVCMOS33 [get_ports clock_rtl]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock_rtl]


#led_tri_o
set_property PACKAGE_PIN M14 [get_ports {led_tri_o[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_tri_o[0]}]


set_property PACKAGE_PIN M15 [get_ports {led_tri_o[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_tri_o[1]}]


set_property PACKAGE_PIN G14 [get_ports {led_tri_o[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_tri_o[2]}]


set_property PACKAGE_PIN D18 [get_ports {led_tri_o[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_tri_o[3]}]
```

3. In the following steps we will Generate Bitstream and launch the SDK.

   (a) Right click in the diagram and select 'Validate Design' to check for any errors in the design. If everything is correct, you have successfully built the hardware platform.

   (b) Right click on your project in the sources panel and select 'Create HDL wrapper'. Select 'Let Vivado manage wrapper and auto update' and click 'OK' to create the top level module for the

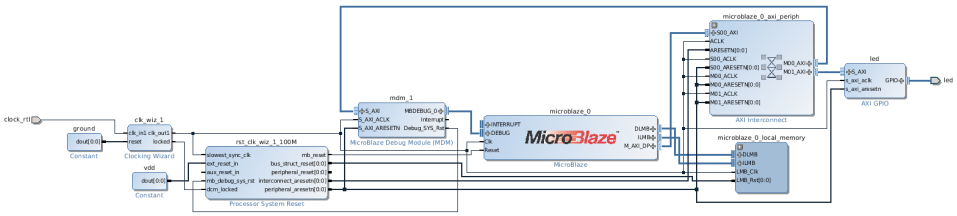

Figure 6: Final Layout

design. Finally, click 'Generate Bitstream' to create the program file. Once this is done, we will be able to export our hardware design and switch over to the SDK to program the FPGA and run the program to manage LEDs.

(c) If any errors are found please rectify them. Once successfully completed, select File → 'Export' → 'Export Hardware'. Check 'Include bitstream' to export the hardware platform.

(d) Now, launch the Software Development Kit(SDK) using 'File' → 'Launch SDK'. Select 'local to project' as the location.

4. In the following steps we will use the SDK to create the software application.

(a) Once the SDK is open. Select 'File' → 'New' → 'Application Project'. Give a project name (eg.counter_sw). Click 'Next' and select the 'Empty Application' and click 'Finish'.

(b) You will now have three files in the 'Project Explorer' panel (counter_sw, counter_sw_bsp and led_wrapper_hw_platform_0).

(c) Using your favorite editor, create a file called 'lab2a.c'. Type the C code shown below in your file and save it in your lab2 directory.

```
#include <xparameters.h>
#include <xgpio.h>
#include <xstatus.h>
#include <xil_printf.h>
/* Definitions */
#define GPIO_DEVICE_ID  XPAR_LED_DEVICE_ID  /* GPIO device that LEDs are connected to */
#define WAIT_VAL 10000000

int  delay (void) ;

int main()
{
  int count;
  int count_masked;
  XGpio leds;
```

```
  int status;

  status=XGpio_Initialize(&leds,GPIO_DEVICE_ID);
  XGpio_SetDataDirection(&leds,1,0x00);
  if (status != XST_SUCCESS) {
      xil_printf('Initialization failed ');
    }
  count=0;
  while(1)
  {
    count_masked=count & 0xF ;
    XGpio_DiscreteWrite(&leds,1,count_masked);
    xil_printf('Value of LEDs = 0x%x\n\r', count_masked);
    delay();
    count++;
  }
  return(0);
}

 int delay(void)
{
  volatile int delay_count=0;
  while(delay_count < WAIT_VAL)
    delay_count++;
  return (0);
}
```

(d) Look through the code you just wrote and try to understand what exactly is going on. Notice we include the 'xparameters.h', 'xstatus.h', and 'xgpio.h' header files. Open up these files from the 'Outline' panel to the right of the window and understand what they provide. At the end of the lab, you will find questions on these files.

(e) In the project explorer, expand 'counter_sw', right click on 'src' and select 'import'. In the import window expand 'General', select 'File System', and click 'Next'. Click 'Browse' and select the folder where you saved the lab2a.c file and click 'OK'. Select the lab2a.c file from the import window and click 'Finish'.

(f) Now we have the software application ready. Connect the FPGA and switch it ON. In the SDK, click 'Xilinx Tools' and select 'Program FPGA'. Next, the 'Program FPGA' window will appear and select 'Program' to program the the bitstream file generated in Vivado on to the FPGA. A warning will appear indicating that there is no PS in the design. PS is short for processing system which represents the ARM Cortex Processor in the ZYBO board. In this lab, the Microblaze processor is used which is implemented completely on the FPGA, and hence it is called a soft processor. Click 'OK' on the message to program the FPGA.

(g) The next step is to run the C application on the system. Click 'Run', and select 'Run Configuration'. In the 'Run Configuration' window , select 'Xilinx C/C++ Application(GDB)'. Click

on the icon which resembles a document with a '+' sign on top left corner of the window to create a configuration file. Name the configuration(eg. led_sw) and in the application tab, select 'Browse' in the Project Name field and select the 'counter_sw'. In the STDIO connection, check 'Connect STDIO to Console'. Select 'JTAG UART' as Port. Click on 'Apply' and select 'Run' to deploy the program to the FPGA.

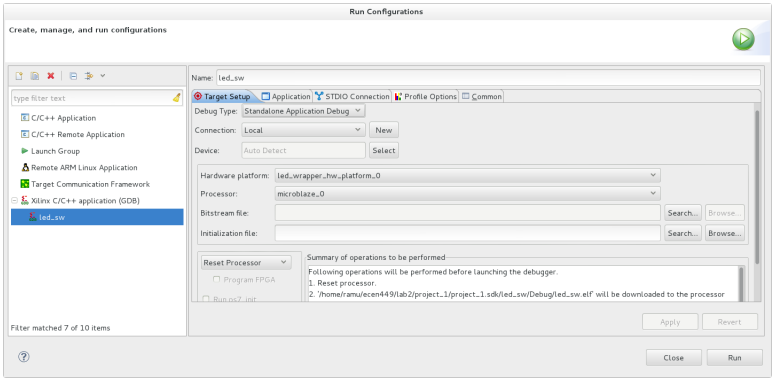

Figure 7: Run Configuration

(h) If everything is correct, you should now be able to see the LEDs glowing in the order from 0 to 15 and the output from the printf statements on the console in the SDK. NOTE: For the purpose of debugging stop the program by entering 'stop' on the XMD console and re-run the program. The XMD console can be found under 'Xilinx Tools'.

Demonstrate your progress to the TA.

Deliverables

1. [2 points.] Demonstrate your work to the TA after downloading the bitstream for the design created using the steps provided in the manual. Add comments to your C code and the modified portion of the XDC file and include them in your lab write-up.

2. [3 points.] Add an 8-bit GPIO IP block to your system and connect the lower 4-bits to the switches and the upper 4-bits to the push buttons on the ZYBO board. Then, develop software to implement the following functionality:

   Your program should keep track of a COUNT value. When the push button 0 is held down, the COUNT should increment at approximately 1 Hz. When push button 1 is held down COUNT should decrement at 1 HZ. When the push button 2 is held down, the status

of the switches should be displayed. When the push button 4 is pressed, COUNT should be displayed on the LEDs. The console should display the current action and LEDs value. Be sure to demo this to your TA.

Hints:

- Skim through the user manual for the ZYBO board and Master XDC file to determine the pin assignments for additional signals. The user manual may be found on the course website.
- Do not forget to add the DIP switches and push buttons to your XDC file before generating the netlist.
- Your source code must detect a change on the push buttons and DIP switches and react accordingly. When a change is detected, print the current action and LEDs value to the terminal window. Also, print to the terminal everytime the LEDs value changes.

3. [5 points.] Correct format including an Introduction, Procedure, Results, and Conclusion.
4. [4 points.] Commented C files.
5. [2 points.] The output of the TCL console from the part 2 demo.
6. [4 points.] Answer the following questions:

   (a) In the first part of the lab, we created a delay function by implementing a counter. The goal was to update the LEDs approximately every second as we did in the previous lab. Compare the count value in this lab to the count value you used as a delay in the previous lab. If they are different, explain why? Can you determine approximately how many clock cycles are required to execute one iteration of the delay for-loop? If so, how many?

   (b) Why is the count variable in our software delay declared as volatile?

   (c) What does the while(1) expression in our code do?

   (d) Compare and contrast this lab with the previous lab. Which implementation do you feel is easier? What are the advantages and disadvantages associated with a purely software implementation such as this when compared to a purely hardware implementation such as the previous lab?