ECEN 749- Microprocessor System Design

Section 603

TA: Mr. Kunal Bharathi

LAB 2

Using the Software Development Kit (SDK)

Date of Performance: 09/12/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

Introduction

This lab will introduce to the Xilinx Vivado Design Block and Xilinx Software Development Toolkit. It will also focus on creating a MicroBlaze processor system using the Xilinx Vivado Design Block. It provides an experience in designing the processor system by adding the General Purpose Input Output Registers (GPIO) in the design. It also introduces the C programming language for coding the processor system.

Design

Procedure: Part 1 Creating MicroBlaze system Processor.

1. Launch Vivado and open new project file.
2. Create new RTL project and skip for 'Add Existing Window' and 'Add Constraints'. The constraints would be added later as a separate file.
3. Then select the Zybo board from the boards tab.
   Set the device properties to the following:
   Device Family: Zynq-7000
   Sub-Family: Zynq-7000
   Package: clg400
   Speed Grade: -1
   And then select the first one among the two devices.
4. Connect the ZYBO board to the PC and switch on. Make sure it is in JTAG mode.
5. In the flow navigator, click on 'Create Block Design' and write name of the design.
6. Then add the MicroBlaze processor from 'Add IP' block
   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)
   After selecting this, clock on 'Run Block Automation'
7. Change the source of primary clock from 'Differential Clock Capable Pin' to 'single ended clock capable pin'.
8. Add GPIO block. In first part, we will be only using output. Hence check all outputs and select GPIO width to '4', indicating 4 LEDs.
9. Replace the reset ports with Constant IPs. On constant IP would be VDD and its constant value is kept one.
10. Connect the constant block to 'ext rst in' of the 'Processor System Reset' because it is 'ACTIVE LOW'.
11. Other constant would be ground and its constant value is kept zero. connect it to the 'reset' port of the 'Clocking Wizard'.
12. The constraints file can be written in a normal text editor like 'Notepad' and then save it with extension of " .xdc".

13. Right click on the constraints tab in the Vivado tool and add the created constraints file.
14. After this step, check for any design errors by clicking on 'Validate Design'.
15. Click on Generate Bitstream. If no errors them go to next step. Otherwise solve the errors first and then proceed.
16. Then click on export and check the 'Include Bitstream'
17. Now launch the SDK.
18. Create new project and select 'Empty Application'
19. Create .c file and store it in the same directory as that of SDK project. The .c file would contain the source code.
20. Import this file in src folder of your project and then click on 'Xilinx Tools' and click on 'Program FPGA'.
21. Then click on Run as Configurations. 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.
22. If the code is correct, then the LEDS will glow in the order from 0 to 15.

C Programming Code

```
#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 0x1000000

int delay(void);

int main() {

    int count;                    //input declaration
    int count_masked;
    XGpio leds;             //for mapping leds on the board with code
    int status;
//checking initialization status
    status = XGpio_Initialize(&leds, GPIO_DEVICE_ID);
    XGpio_SetDataDirection(&leds, 1, 0x00);
    if(status != XST_SUCCESS) {
        xil_printf("Initialization failed");
    }
    count = 0;              //initiating count to 0
    while(1) {              //ensures that loop runs continuously
        count_masked = count & 0xF;    //output is limited to 15
        XGpio_DiscreteWrite(&leds, 1, count_masked);
        xil_printf("LEDs = 0x%x \n\r", count_masked);
        delay();
```

```
        count++;
    }
    return (0);
}
//delay which acts a clock divider in Verilog
int delay(void) {
    volatile int delay_count = 0;
    while(delay_count < WAIT_VAL)
        delay_count++;
    return(0);
}
```

Constraints file.

```
## clock_rtl
set_property PACKAGE_PIN K17 [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]}]
```

Procedure: Part 2

In this experiment, we have four cases that are triggered by each of the pushbutton on the ZYBO-Z7-10 board. They are:

a. When the push button 0 is held down, the COUNT should increment at approximately 1 Hz.
b. When push button 1 is held down COUNT should decrement at 1 HZ.
c. When the push button 2 is held down, the status of the switches should be displayed.
d. When the push button 3 is pressed, COUNT should be displayed on the LEDs. The console should display the current action and LEDs value.

1. In this part, everything is similar to Part 1 apart from that here two GPIOs are needed, One for output and other for inputs (Switches and Pushbuttons).
2. The second GPIO will be All Inputs and GPIO width=8.
3. Refer steps 1 to 22 from part 1.
4. Then write the program in .c file and upload the code. Then check the output on console and the LEDs.

C Programming code

```
#include <xparameters.h>
#include <xgpio.h>
#include <xstatus.h>
#include <xil_printf.h>

/* Definitions */
#define GPIO_DEVICE_ID_0 XPAR_LED_DEVICE_ID   /* GPIO device that
LEDs are connected to */
#define GPIO_DEVICE_ID_1 XPAR_AXI_GPIO_0_DEVICE_ID /*GPIO device
that switches are connected to*/
#define WAIT_VAL 0x1000000

int delay(void);

int main() {

    int count;                    //variables declaration
    int count_masked;
    XGpio leds;
    XGpio switches;
    //int status;
    int switchstatus; //receives status of switches from the board
    int status_switch;//for initialization status check
    int status_led;
    int buttonstatus;/
    int switchstat;

    status_led = XGpio_Initialize(&leds, GPIO_DEVICE_ID_0);
    XGpio_SetDataDirection(&leds, 1, 0x00);
    if(status_led != XST_SUCCESS) {
        xil_printf(" LEDS Initialization failed");
    }

    status_switch = XGpio_Initialize(&switches, GPIO_DEVICE_ID_1);
        XGpio_SetDataDirection(&switches, 1, 0x01);
        if(status_switch != XST_SUCCESS) {
            xil_printf("Switches Initialization failed");
        }


    count = 0;
    count_masked=0;
    switchstatus=0;
while (1)
{
    switchstatus=XGpio_DiscreteRead(&switches,1);

    buttonstatus=switchstatus & 0X0F; /*masking to receive only
lower 4 bits*/


    switchstat=switchstatus & 0XF0; /*to receive only upper four
bits*/
```

```
    switchstat=switchstat>>4;//shifting bits from right to left
    switch (buttonstatus)
    {
    case 1://UP COUNTER
       count_masked = count & 0xF;
             // XGpio_DiscreteWrite(&leds, 1, count_masked);
              xil_printf("LEDs = 0x%x \n\r", count_masked);
              delay();
              count++;
              break;
    case 2://DOWN COUNTER
       count_masked = count & 0xF;
              //XGpio_DiscreteWrite(&leds, 1, count_masked);
              xil_printf("LEDs = 0x%x \n\r", count_masked);
              delay();
              count--;
              break;
   //status of switch
    case 4:
      XGpio_DiscreteWrite(&leds, 1, switchstat);
      if(switchstat==4)
            switchstat=3;
      if(switchstat==8)
            switchstat=4;
      xil_printf("Value of Counter is 0x%x \n\r", count_masked);
      xil_printf("Switch status= 0x%x \n\r", switchstat);

      delay();
      break;
   //status of count
    case 8:
      count_masked = count & 0xF;
              XGpio_DiscreteWrite(&leds, 1, count_masked);
      xil_printf("Value of Counter is 0x%x \n\r", count_masked);
      xil_printf("Switch status= 0x%x \n\r", switchstat);

              delay();
         break;
    default:
      ;
    }
}
return (0);
}

//delay function acting as a clock divider
int delay(void) {
    volatile int delay_count = 0;
    while(delay_count < WAIT_VAL)
        delay_count++;
    return(0);
}
```

Constraints.

```
## clock_rtl
set_property PACKAGE_PIN K17 [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]}]

## pushbutton
set_property PACKAGE_PIN K18 [get_ports {gpio_rtl_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[0]}]

set_property PACKAGE_PIN P16 [get_ports {gpio_rtl_tri_i[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[1]}]

set_property PACKAGE_PIN K19 [get_ports {gpio_rtl_tri_i[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[2]}]

set_property PACKAGE_PIN Y16 [get_ports {gpio_rtl_tri_i[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[3]}]

##switches
set_property PACKAGE_PIN G15 [get_ports {gpio_rtl_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[4]}]

set_property PACKAGE_PIN P15 [get_ports {gpio_rtl_tri_i[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[5]}]

set_property PACKAGE_PIN W13 [get_ports {gpio_rtl_tri_i[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[6]}]

set_property PACKAGE_PIN T16 [get_ports {gpio_rtl_tri_i[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {gpio_rtl_tri_i[7]}]
```

Output of Console:

```
<terminated> led_sw [Xilinx C/C++ application (GDB)]
LEDs = 0x0
LEDs = 0x1
LEDs = 0x2
LEDs = 0x3
LEDs = 0x4
LEDs = 0x5
LEDs = 0x6
LEDs = 0x7
LEDs = 0x8
LEDs = 0x9
LEDs = 0xA
LEDs = 0xB
LEDs = 0xC
LEDs = 0xD
LEDs = 0xE
LEDs = 0xF
LEDs = 0x0
LEDs = 0x1
```

Problems | Tasks | Console | Properties | Terminal

```
<terminated> counterlab2b [Xilinx C/C++ application (GDB)] /home/gr
Value of Counter is 0x0
Switch Status = 0x0
Value of Counter is 0x0
Switch Status = 0x0
Value of Counter is 0x0
Switch Status = 0x0
Value of Counter is 0x1
Switch Status = 0x0
Value of Counter is 0x2
Switch Status = 0x0
Value of Counter is 0x3
Switch Status = 0x0
Value of Counter is 0x4
Switch Status = 0x0
Value of Counter is 0x5
Switch Status = 0x4
Value of Counter is 0x5
Switch Status = 0x4
Value of Counter is 0x5
Switch Status = 0x4
Value of Counter is 0x5
Switch Status = 0x4
```

Result:

In the first part of lab, the major task was preparing the design circuit using MicroBlaze, as a small mistake in configurating the components would create a problem at a later stage. The important part was understanding the library files such as Xparameters, XGpio, etc since there were many instructions and variables using these libraries. While performing second part of the experiment, it provided a clear understanding of the program. Mapping of inputs and outputs of Zybo board with that of the MicroBlaze Soft IP system was also learnt.

Conclusion:

This lab served its purpose of introducing Xilinx Software development kit. Also it provided hands on experience of designing a MicroBlaze processor system design and programming it with C language. New features of Xilinx SDK were also learnt.

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?

Ans. The value in the previous lab and this lab were different. In the previous lab the value was 124,999,999 whereas in this lab it is 1000000. This is because of the difference between internal clock frequency of Zybo Z7-10 and MicroBlaze Processor. Zybo z7-10 had an internal clock frequency of 125 Mhz and MicroBlaze has an internal frequency of 100 Mhz. The number of clock cycles needed for executing an instruction depends on the microarchitecture of the processor and pipelining used. Hence if we assume that it takes one clock cycle for executing a single instruction, the delay while loop would require 2*10^6 instructions, along with 2 instructions for initializing the count.

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

Ans. The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler. Hence declaring count as volatile would prevent it from any optimization and any undesired changes in output.

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

Ans. The while (1) in our code uses causes the statements inside the loop to run continuously for infinite time.

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?

Ans. The implementation of hardware in previous lab was much easier than software implementation, as we already had the hardware and only the bitstream had to be generated. However, in practical and industrial scenario software implementation is more advantageous than hardware implementation. In Software implementation, we can add or remove components and configure them the way we want and run the program to check if it matches our requirement or what further modification are needed. But on the other hand, the same flexibility cannot be obtained using hardware implementation.