Lab #2: SDK

Khanh Nguyen

UIN# 525000335

ECEN 449– 503

Date: February 2, 2017

INTRODUCTION:

The purpose of the lab is to get us the first touch to get used with Software Development Kit (SDK). We learned how to create a Microblaze processor system and design a software to control the LEDs and switches.

PROCEDURE:

1/ Create a new block design and add the MicroBlaze processor to the design. Run block automation.

2/ Configure the system clock (single ended clock capable) of 100MHz.

3/ Add GPIO block and configure it to be the output of LEDs.

4/ Connect the components automatically by running connection automation.

5/ Connect High and Low signal to reset pins of the processor and the clock.

6/ Add the contraints file for the LEDs.

7/ Generate bitstream and create a HDL wrapper and switch to SDK

8/ Create and import the provided C code to the SDK

9/ Program the FPGA through SDK and configure to run it.

10/ Add a new 8-bit GPIO IP block as inputs and use it to create a “count” program that can increment, decrement and keep track of the count value.

RESULT:

I created a new 8-it GPIO blocks for the inputs of the counter program. I went through the header file xparameters.h to obtain the address of the block and use it in the SetDataDirection function. The logic is almost the same as lab 1, however, this time I used C.

CONCLUSION:

I learned how to use the SDK to create a MicroBlaze system and use GPIO blocks to create inputs and output for the system. I also learned to how to implement the FPGA using C.

The process of reading through the header files and the C code helps me understand the syntax and the use of the new functions: SetDataDirection, DiscreteWrite, DiscreteRead, etc in SDK.

VERILOG CODES:

PART1:

C file

```
#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 1000000000

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);//Assign Gpio block as output in channel 1
	if (status !=XST_SUCCESS){//check if the initialization success
		xil_printf("Intialization failed");
	}
	count=0;
	while (1) //infinity loop to keep the program running
```

```
	{
		count_masked=count & 0xF; //0xF = 1111 in binary which are the 4 bits of LEDs
		XGpio_DiscreteWrite(&leds,1,count_masked);//write the value of count_masked
to LEDs
		xil_printf("Value of LEDs = 0x%x \n \r",count_masked);// print value of LEDs
		delay(); // call delay function
		count++; //increase count
	}
	return(0);
}

int delay(void)
{
	volatile int delay_count=0; //initialize delay
	while (delay_count< WAIT_VAL) // increase delay time ulti it reaches the wait time
		delay_count++;
	return (0);
}
```

XDC file:

```
#Initialize 100MHz clock
#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]

#Initialize LEDs
#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]}]
```

PART 2:

C file:

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

#define LED XPAR_LED_DEVICE_ID
#define SWITCH XPAR_SWITCH_DEVICE_ID
// Search the xparameters.h file for the GPIO block addresses
#define WAIT_VAL 1000000000

int delay (void) ;

int main()
{
        int count;
```

```
int count_masked;
XGpio leds;
XGpio switches;
int status1, status2;
int val,val1,val2;
status1 =XGpio_Initialize(&leds, LED);
XGpio_SetDataDirection(&leds,1,0x00); // Initialize GPIO block 1 as output
if(status1 != XST_SUCCESS) {//check for errors
	xil_printf("Initialization failed");
}

status2 =XGpio_Initialize(&switches, SWITCH);
XGpio_SetDataDirection(&switches,1,0xFF);//GPIO block 2 as input (switches)
if(status2 != XST_SUCCESS) {//check for errors
	xil_printf("Initialization failed");
}
count =0;
while (1)
{
	val = XGpio_DiscreteRead(&switches,1); // read values from block 2
	val1 = val & 0x0F;//lower 4 bit value

	if(val1 == 0x1) { //if switch is 1, increment count
		count++;
		count_masked = count & 0xF;
		xil_printf("Value of COUNT = 0x%x \n\r", count_masked);
	}
	else if(val1 == 0x2) { // if switch is 2, decrement count
```

```
                count--;
                count_masked = count & 0xF;
                xil_printf("Value of COUNT = 0x%x \n\r", count_masked);
        }
        else if(val1 == 0x8) { // if switch is 8, display count
                count_masked = count & 0xF;
                XGpio_DiscreteWrite(&leds,1,count_masked);              //Write to
LEDs
                xil_printf("Value of COUNT = 0x%x \n\r", count_masked);
        }
        else if(val1 == 0x4) { //if switch is 4, display the status of switches
                val2 = XGpio_DiscreteRead(&switches,1) & 0xF0;// 0xF0 is  11110000
(upper 4 bits)
                val2 = val2/16;
                XGpio_DiscreteWrite(&leds,1,val2);//write to LEDs
                xil_printf("Value of SWITCHES = 0x%x \n\r",val2);
        }
        delay();
    }
    return(0);
}

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

XDC File:

```
#clock_rt1
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]}]
//Use the variable name generated in the design tab
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]}]


#switch_tri_i
set_property PACKAGE_PIN R18 [ get_ports {switch_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {switch_tri_i[0]}]

set_property PACKAGE_PIN P16 [ get_ports {switch_tri_i[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {switch_tri_i[1]}]
```

set_property PACKAGE_PIN V16 [get_ports {switch_tri_i[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[2]}]

set_property PACKAGE_PIN Y16 [get_ports {switch_tri_i[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[3]}]

set_property PACKAGE_PIN G15 [get_ports {switch_tri_i[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[4]}]

set_property PACKAGE_PIN P15 [get_ports {switch_tri_i[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[5]}]

set_property PACKAGE_PIN W13 [get_ports {switch_tri_i[6]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[6]}]

set_property PACKAGE_PIN T16 [get_ports {switch_tri_i[7]}]

set_property IOSTANDARD LVCMOS33 [get_ports {switch_tri_i[7]}]

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?

In previous lab the clock rate was 125MHz, and since I have to reverse the clock every 0.5s, so 40,000,000 clock cycles will be required for 0.5s.

In this lab, I don't have to reverse the clock so 100,000,000 clock cycles will be required for 1s

100MHz=10E-8s => (10E-8s)*(10E8)=1s

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

It is declared like that because the delay can be changed and the timescale per clock cycle depends on the processor.

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

It is the infinity loop, the loop will run until user aborts the program.

(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?

It was my first time implement the FPGA board using SDK and C. Therefore, I still feel Verilog is a little bit easier. Software implementation is little bit harder (due to some driver problems and unknown errors, I had to recreate the whole design three times). However, using SDK and C, I was able to check the output from both software and FPGA board. Pure software implementation can reduce the cost and the risk of hardware failure, while hardware implementation rely a lot on the quality of hardware used.