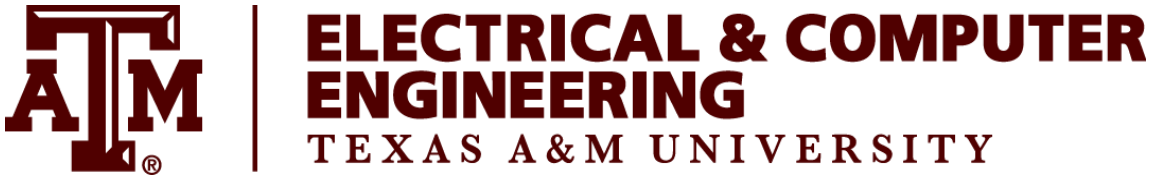


ECEN 449: Microprocessor System Design

Lab 2: Using the Software Development Kit (SDK)

Kylan Lewis

UIN: 719001131

ECEN 449 -504

TA: Ashwin Ashokan

Date: 9/25/2020

Introduction:

The purpose of this lab is to learn how to do the general procedures of implementing the microprocessors and GPIO using block diagrams in vivado and using the SDK to run code on the FPGA board.

Procedure:

Part 1:

1. Put in microprocessor and GPIO both into a block diagram
2. Then I created an HDL wrapper
3. Generate bitstream
4. Opened up the SDK
5. Imported and edited the c code
6. Programmed the FPGA
7. Test the functionality

Part 2:

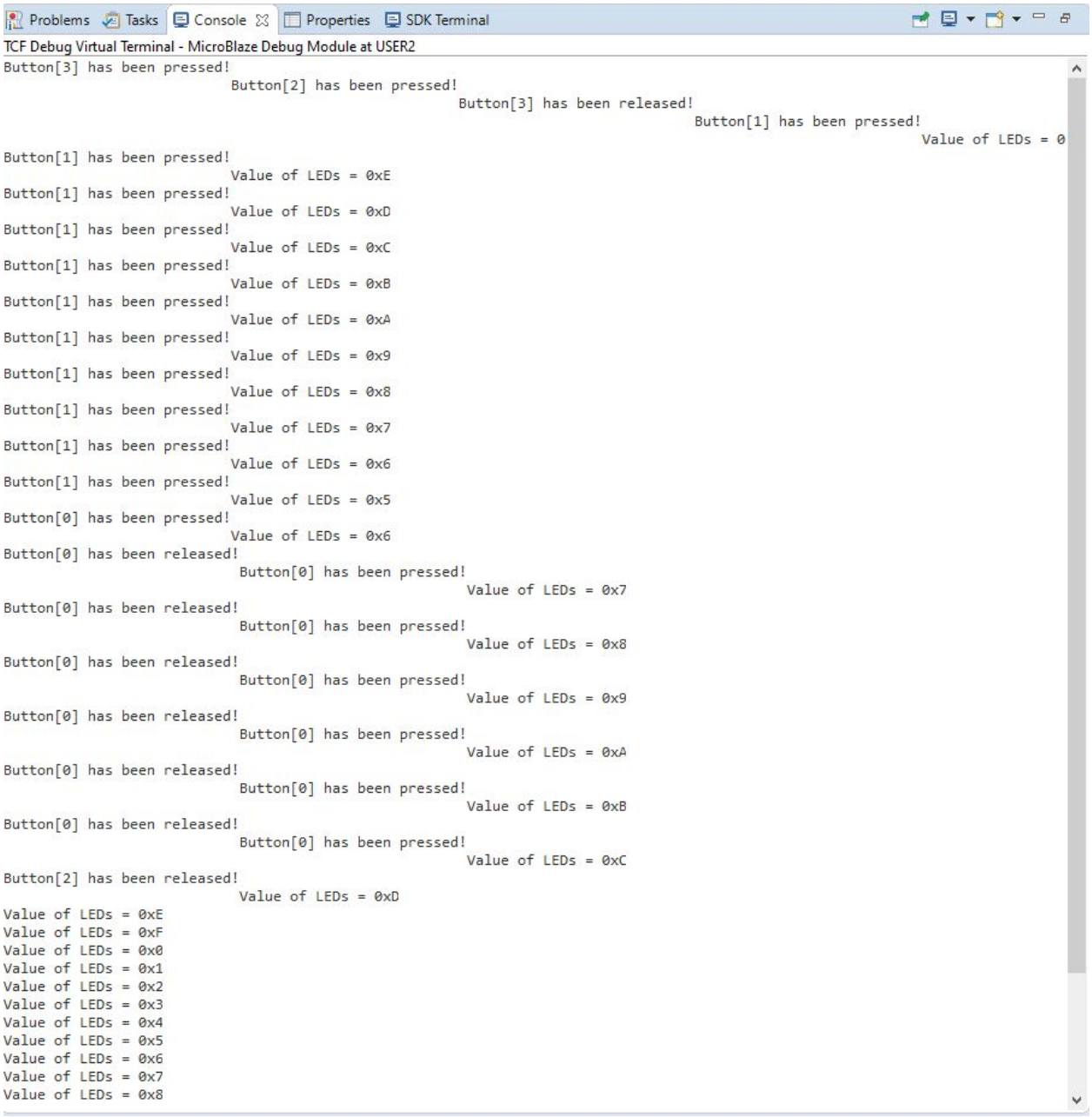
1. Did the same thing as part one from steps 1-4 but also included a 8bit GPIO into the block diagram
2. Created new c code and programmed the FPGA
3. Test functionality

Results:

The results were predictable. In part 2, the first 2 buttons increased and decreased the count. The next 2 buttons, one of them showed the current count and the other showed the switched LED positions value. 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 C was used.

Outputs:


TCF Debug Virtual Terminal - MicroBlaze Debug Module at USER2
Value of LEDs = 0x0
Value of LEDs = 0x1
Value of LEDs = 0x2
Value of LEDs = 0x3
Value of LEDs = 0x4
Value of LEDs = 0x5
Value of LEDs = 0x6
Value of LEDs = 0x7
Value of LEDs = 0x8
Value of LEDs = 0x9
Value of LEDs = 0xA
Value of LEDs = 0xB


Problems
Tasks
Console
Properties
SDK Terminal
TCF Debug Virtual Terminal - MicroBlaze Debug Module at USER2
Button[3] has been pressed!
Button[2] has been pressed!
Button[3] has been released!
Button[1] has been pressed!
Value of LEDs = 0
Button[1] has been pressed!
Value of LEDs = 0xE
Button[1] has been pressed!
Value of LEDs = 0xD
Button[1] has been pressed!
Value of LEDs = 0xC
Button[1] has been pressed!
Value of LEDs = 0xB
Button[1] has been pressed!
Value of LEDs = 0xA
Button[1] has been pressed!
Value of LEDs = 0x9
Button[1] has been pressed!
Value of LEDs = 0x8
Button[1] has been pressed!
Value of LEDs = 0x7
Button[1] has been pressed!
Value of LEDs = 0x6
Button[1] has been pressed!
Value of LEDs = 0x5
Button[0] has been pressed!
Value of LEDs = 0x6
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0x7
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0x8
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0x9
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0xA
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0xB
Button[0] has been released!
Button[0] has been pressed!
Value of LEDs = 0xC
Button[2] has been released!
Value of LEDs = 0xD
Value of LEDs = 0xE
Value of LEDs = 0xF
Value of LEDs = 0x0
Value of LEDs = 0x1
Value of LEDs = 0x2
Value of LEDs = 0x3
Value of LEDs = 0x4
Value of LEDs = 0x5
Value of LEDs = 0x6
Value of LEDs = 0x7
Value of LEDs = 0x8

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 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.

Post-Lab Questions :

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?

The count I used for this lab compared to lab 1 both update every the second even though the counts are different. In lab 1 I used 150000000 because the clock given for the FPGA is 150 MHz. The count I used for this lab was 10000000, this is because the system updates at 1 Hz. 2.

Why is the count variable in our software delay declared as volatile?
Because the count variable can change on its own at any time.

What does the while(1) expression in our code do?
The while loop is an infinite loop that runs until an interrupt/abort happens.

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?

Lab 1 was easier than lab 2 because of how little steps we had to take doing the hardware implementation. This is only because we had to do simple tasks for the first 2 labs. As the labs get harder and harder the more likely software implementation will be needed.

Code:

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

#define WAIT_VAL 0x1000000
#define GPIO_DEVICE XPAR_LED_DEVICE_ID

int delay(void);

int main(){

	int count;
	int count_masked;
	XGpio led; //Hardware io
	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) //infinite loop to keep circuit always executing
	{
		count_masked = count & 0xf;//get lower 4 bits of count, so that count masked rolls over
every 16 counts
		XGpio_DiscreteWrite(&led,1,count_masked);//write count masked to leds
		xil_printf("Value of LEDs = 0x%x\n\r",count_masked); //Print count to console
		delay();
		count++;
	}

	return (0);
}

int delay (void)
{
	volatile int delay_count=0; // volatile prevents compiler optimization
	while(delay_count<WAIT_VAL)
		delay_count++;// each iteration of while loop is 2 clock cycles
	return(0);
}
```

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

//Definitions
#define GPIO_DEVICE_ID_LEDS XPAR_LED_DEVICE_ID //device IDs
#define GPIO_DEVICE_ID_SWB XPAR_BTN_SW_DEVICE_ID
#define WAIT_VAL 10000000 //100 MHz clock input (we need to divide it by 10 mil to get 1 Hz)
#define boolean _Bool //boolean is more pleasant to type than _Bool
#define false 0 //vivado requires this in order to compile
#define true 1

int delay(void) { //slows the 100 MHz clock down to 1 MHz
	volatile int delay_count = 0; //tells the compiler that this value may change at any time without
any action taken
	while (delay_count < WAIT_VAL) {
		delay_count++;
	}
	return (0);
}

int main() {
	int count = 0; //actual count value (may be greater than 15)
	int count_masked = 0; //count_masked will always be between 0 and 15
	XGpio leds; //LEDs
	XGpio swb; //switches/buttons
	int statuso; //verifying satisfactory status of the output port

	statuso = XGpio_Initialize(&leds, GPIO_DEVICE_ID_LEDS);
	XGpio_SetDataDirection(&leds, 1, 0); //0 is for output
	if (statuso != XST_SUCCESS) {
		xil_printf("Initialization failed (LEDs)");
	}

	XGpio_Initialize(&swb, GPIO_DEVICE_ID_SWB);
	XGpio_SetDataDirection(&swb, 1, 1);//1 is for input
	int switchValue = 0;

	//this is how we keep track of whether variables change
	boolean b0 = false;
	boolean b1 = false;
```

```
	boolean b2 = false;
	boolean b3 = false;

	for (;;) { //this loop will go until the user terminates the program manually
		boolean button0 = (0x01 & XGpio_DiscreteRead(&swb, 1) == 0x01) //button 0 is
currently pressed (overrides button 1 being pressed)
			&& !((XGpio_DiscreteRead(&swb, 1) & 0x04) == 0x04) //NOT button2
			&& !((XGpio_DiscreteRead(&swb, 1) & 0x08) == 0x08); //NOT button3

		boolean button1 = ((XGpio_DiscreteRead(&swb, 1) & 0x02) == 0x02) //button 1 is being
pressed
			&& !((XGpio_DiscreteRead(&swb, 1) & 0x04) == 0x04) //NOT button2
			&& !((XGpio_DiscreteRead(&swb, 1) & 0x08) == 0x08); //NOT button3

		boolean button2 = ((XGpio_DiscreteRead(&swb, 1) & 0x04) == 0x04); //button 2 is
being pressed

		boolean button3 = ((XGpio_DiscreteRead(&swb, 1) & 0x08) == 0x08); //button 3 is
being pressed

		if (!button2 && !button3) XGpio_DiscreteWrite(&leds, 1, 0); //turn off leds when no
relevant buttons are pressed

		if (button0) { //bitwise AND with 1 to find value of last bit
			if (!b0) { xil_printf("Button[0] has been pressed!\n"); b0 = true; }
			if (b1) { b1 = false; xil_printf("Button[1] has been released!\n"); }
			if (b2) { b2 = false; xil_printf("Button[2] has been released!\n"); }
			if (b3) { b3 = false; xil_printf("Button[3] has been released!\n"); }
			count++;
			count_masked = count & 0xF;
			xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
			//XGpio_DiscreteWrite(&leds, 1, count_masked);
			delay();
		}
		else if (button1) {
			count--;
			//checking changes of values
			if (!b1) { b1 = true; xil_printf("Button[1] has been pressed!\n"); }
			if (b0) { b0 = false; xil_printf("Button[0] has been released!\n"); }
			if (b2) { b2 = false; xil_printf("Button[2] has been released!\n");  }
			if (b3) { b3 = false; xil_printf("Button[3] has been released!\n");  }

			count_masked = count & 0xF;
			xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
			//XGpio_DiscreteWrite(&leds, 1, count_masked);
			delay();
		}

		else if (button2) { /*button 2 pressed (show switches)*/
			if (b1) { b1 = false; xil_printf("Button[1] has been released!\n"); }
			if (b0) { b0 = false; xil_printf("Button[0] has been released!\n"); }
			if (!b2) { b2 = true; xil_printf("Button[2] has been pressed!\n"); }
			if (b3) { b3 = false; xil_printf("Button[3] has been released!\n");}
```

```
                        if ((XGpio_DiscreteRead(&swb, 1) & 0xF0) != switchValue) {
                                switchValue = XGpio_DiscreteRead(&swb, 1) & 0xF0;
                                xil_printf("You moved a switch! Switch Value: %d\n", switchValue >>
4);
                        }
                        XGpio_DiscreteWrite(&leds, 1, switchValue >> 4);
                        //display leds with switch values

                        if ((XGpio_DiscreteRead(&swb, 1) & 0x01) == 0x01) { /*button2 is pressed
AND button0 is pressed*/
                                if (!b0) { b0 = true; xil_printf("Button[0] has been pressed!\n"); }
                                count++;
                                count_masked = count & 0xF;
                                xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
                                delay();
                        }
                        else if ((XGpio_DiscreteRead(&swb, 1) & 0x02) == 0x02) { /*button2 is pressed
AND button1 is pressed*/
                                if (!b1) { b2 = true; xil_printf("Button[1] has been pressed!\n"); }
                                count--;
                                count_masked = count & 0xF;
                                xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
                                delay();
                        }

                }
                else if (button3) {
                        //same check
                        if (b1) { b1 = false; xil_printf("Button[1] has been released!\n"); }
                        if (b0) { b0 = false; xil_printf("Button[0] has been released!\n"); }
                        if (b2) { b2 = false; xil_printf("Button[2] has been released!\n");}
                        if (!b3) { b3 = true; xil_printf("Button[3] has been pressed!\n"); }

                        if ((XGpio_DiscreteRead(&swb, 1) & 0x01) == 0x01) { //if button 3 AND button
0
                                count++;
                                count_masked = count & 0xF;
                                XGpio_DiscreteWrite(&leds, 1, count_masked);
                                xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
                                delay();
                        }
                        else if ((XGpio_DiscreteRead(&swb, 1) & 0x02) == 0x02) { //if button 3 AND
button 1
                                if (!b1) { b1 = true; xil_printf("Button[1] has been pressed!\n"); }
                                count--;
                                count_masked = count & 0xF;
                                XGpio_DiscreteWrite(&leds, 1, count_masked);
                                xil_printf("Value of LEDs = 0x%x\n\r", count_masked);
                                delay();
                        }
                        else { //just button 3 is pressed; we simply display the current count value
                                XGpio_DiscreteWrite(&leds, 1, count_masked);
```

```
                    }

                }
            }
}
```

```
#clock_100MHz
set_property PACKAGE_PIN K17 [get_ports clk_100MHz]
set_property IOSTANDARD LVCMOS33 [get_ports clk_100MHz]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk_100MHz]


#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]}]

#sw
set_property PACKAGE_PIN G15 [get_ports {btn_sw_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[0]}]

set_property PACKAGE_PIN P15 [get_ports {btn_sw_tri_i[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[1]}]

set_property PACKAGE_PIN W13 [get_ports {btn_sw_tri_i[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[2]}]

set_property PACKAGE_PIN T16 [get_ports {btn_sw_tri_i[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[3]}]

#btn
set_property PACKAGE_PIN K18 [get_ports {btn_sw_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[4]}]

set_property PACKAGE_PIN P16 [get_ports {btn_sw_tri_i[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[5]}]
```

```
set_property PACKAGE_PIN K19 [get_ports {btn_sw_tri_i[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[6]}]

set_property PACKAGE_PIN Y16 [get_ports {btn_sw_tri_i[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {btn_sw_tri_i[7]}]
```