ECEN 449 Lab Report 2

Philip Smith - 624002014 (Sec. 511)

February 3, 2020

Introduction

The focus of Lab 2 was to become familiar with several new concepts included in Vivado. The Lab focuses around burning the MicroBlaze processor to the FPGA board. MicroBlaze is an open source "soft microprocessor[1]," that is, a CPU architecture designed to be programmed entirely on a FPGA. In addition the lab was designed to introduce us to the concept of IP (Intellectual Property) blocks within Vivado.

The lab consisted of two section:

1. Burn the MicroBlaze MCU onto the board and execute a test program
2. Create a second program with additional functionality

[1] MicroBlaze Architecture: `https://www.xilinx.com/products/design-tools/microblaze.html`

Procedure

The first step is to create a block diagram and import all of the IP blocks we need. These are:

1. MicroBlaze : The microcontroller used to run the programs
2. GPIO block : Used to interface with the LEDs, Switches, and Buttons via GPIO bus

After creating this, we synthesize, implement, and generate the bitstream. This is exported and the SDK is opened in the same path. The SDK allows us to use JTAG over serial to debug the program as it is in the process of executing, as well as to compile the program for the MicroBlaze controller.

The first program simply uses a counter and displays the count on the 4 on-board LEDs.

The second program can take user input and display an up/down count as well as other functionality that I had to program myself based on the source of the first problem that was given to us.

Results

The only issue I ran into was an incorrect copy of the XDC file which led to pins missing definitions. The logs immediately assisted me in resolving the issue. Also, I had initially forgotten to resize the GPIO bus. This resulted in the LEDs not lighting up despite the program working. My TA assisted me in finding a solution.

The second program I wrote was easy and ran correctly the first time with all functions operational.

Conclusions

TODO

Questions

1. Lab01 saw a clock division counter of 2^{32}. This lab uses a counter of only 10 million. The clocking wizard was setup to provide the system with a 100MHz clock. This means the final frequency is approximately $100MHz/10000000 = 10Hz$, meaning the entire loop takes approximately 10 cycles to complete.

2. The count variable is instantiated with the volatile directive so as to prevent the compiler from causing issues with it should it optimize it in some manner. In short, it prevents the compiler from optimizing the variable which may have unintended consequences.

3. The while(1) expression is an implementation of an infinite loop.

4. I believe that this implementation, while it takes longer to setup and syntheisize, is overall easier for several reaosons. It is easier to debug via JTAG, and it is easier to be more explicit in what you desire from a program. Verilog has a lot of pitfalls and traps as seen in our lecture portion that a higher level language like C can help a programmer avoid. In addition it should be noted that compiling and uploading a program is much faster than synthesizing a program with similar functionality in Verilog, however, I do not know if this scales up or down.

Appendix A - lab2a.c

The following is the code for lab2a.c:

```
#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 //wait value used to divide the clock

int delay (void);

int main() {
        int count; //how many times we go through the while loop in total
        int count_masked; //the first 4 bits (what is shown on the LEDs)
        XGpio leds; //GPIO output led port
        int status; //tells us if we successfully initialize the GPIO device

        status = XGpio_Initialize(&leds, GPIO_DEVICE_ID);
        XGpio_SetDataDirection(&leds, 1, 0x00); //data direction (1 for input, 0 for output)
        if (status != XST_SUCCESS) {
                xil_printf("Initialization failed");
        }
        count = 0;//define count to be 0 before the loop
        while(1) { //the loop repeats until the user manually terminates
                count_masked = count & 0xF; //count bitwise and with 15 to only display first 4 bits on LED
                XGpio_DiscreteWrite(&leds, 1, count_masked); //DiscreteWrite is used to give values to outp
                xil_printf("Value of LEDs = 0x%x\n\r", count_masked); //console output as a checkpoint
                delay(); //delay by 10,000,000 cycles through while loop
                count++; //increment count
        }
        return(0);
}

int delay(void) {
        volatile int delay_count = 0; //volatile prevents compiler from applying optimizations to changing
        while(delay_count < WAIT_VAL)
                delay_count++; //keep looping until we reach WAIT_VAL (10,000,000 cycles define above)
        return (0);
}
```

Appendix B - lab2b.c

The following is the code for lab2b.c:

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

//Definitions
#define GPIO_DEVICE_ID_LED XPAR_LED_DEVICE_ID //From xparams.h include
#define GPIO_DEVICE_ID_SWS XPAR_SWS_DEVICE_ID // ^
#define WAIT_VAL 10000000 //wait value used to divide the clock

int delay (void);

int main(){

        int count; //how many times we go through the while loop in total
        int count_masked; //the first 4 bits (what is shown on the LEDs)
        XGpio leds; //GPIO output led port
        XGpio sws;
        int status; //tells us if we successfully initialize the GPIO device
        int switches;

        //Init LED GPIO
        status = XGpio_Initialize(&leds, GPIO_DEVICE_ID_LED);
        XGpio_SetDataDirection(&leds, 1, 0x00); //data direction (1 for input, 0 for output)
        if (status != XST_SUCCESS) { xil_printf("Initialization failed for LEDs\n\r"); }

        //Init SWITCHES and BUTTONS GPIO
        status = XGpio_Initialize(&sws, GPIO_DEVICE_ID_SWS);
        XGpio_SetDataDirection(&sws, 1, 0x00); //data direction (1 for input, 0 for output)
        if (status != XST_SUCCESS) { xil_printf("Initialization failed for INPUT\n\r"); }


        count = 0; //counter variable set to 0 before program begins looping

        while(1){
                //GPIO map:
                //sw3 sw2 sw1 sw0        b3 b2 b1 b0
                //7   6   5   4     3  2  1  0
                _Bool b0 = ( (0x01 & XGpio_DiscreteRead(&sws,1)) == 0x01); //Button 1 status
                _Bool b1 = ( (0x02 & XGpio_DiscreteRead(&sws,1)) == 0x02); //Button 2 status
                _Bool b2 = ( (0x04 & XGpio_DiscreteRead(&sws,1)) == 0x04); //Button 3 status
                _Bool b3 = ( (0x08 & XGpio_DiscreteRead(&sws,1)) == 0x08); //Button 4 status

                if (b0) {
                        XGpio_DiscreteWrite(&leds,1,0x00); //disable LEDs
                        count++; //increment count
                        xil_printf("USER: BUTTON 0 FUNCTION: INCREMENT COUNT LEDS: OFFLINE\n\r");
                        delay();
                }

                if (b1) {
                        XGpio_DiscreteWrite(&leds,1,0x00); //disable LEDs
                        count--; //decrement count
                        xil_printf("USER: BUTTON 1 FUNCTION: DECREMENT COUNT LEDS: OFFLINE\n\r");
                        delay();
                }

                if(b2){
                        switches = (0xF0 & XGpio_DiscreteRead(&sws,1)) >> 4; //Get switch mask and shift
                        XGpio_DiscreteWrite(&leds,1,switches); //Write switches to LEDs
```

```
                        xil_printf("USER: BUTTON 2 FUNCTION: SHOW SWITCHES   LEDS: 0x%x\n\r", switches);
                        delay();
                }

                if(b3){
                        count_masked = count & 0xF; //get low nibble of count
                        XGpio_DiscreteWrite(&leds,1,count_masked); //write count to LEDs
                        xil_printf("USER: BUTTON 1 FUNCTION: DECREMENT COUNT LEDS: 0x%x\n\r", count_masked)
                        delay();
                }
        }
}

int delay(void) {
        volatile int delay_count = 0; //volatile prevents compiler from applying optimizations
        while(delay_count < WAIT_VAL)
                delay_count++; //keep looping until we reach WAIT_VAL (10,000,000 cycles define above)
        return (0);
}
```

Appendix C - led.xdc

The following is the code for the led.xdc constraints file (for both sections):

```
#CLOCK
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 ]

#LEDS 0-3
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]}]

#SWITCHES (BITS 0 - 3)
set_property PACKAGE_PIN G15 [ get_ports { buttons_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports { buttons_tri_i[0]}]

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

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

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


#BUTTONS (BITS 4 - 7)
set_property PACKAGE_PIN R18 [ get_ports { buttons_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [ get_ports { buttons_tri_i[4]}]

set_property PACKAGE_PIN P16 [ get_ports { buttons_tri_i[5]}]
set_property IOSTANDARD LVCMOS33 [ get_ports { buttons_tri_i[5]}]

set_property PACKAGE_PIN V16 [ get_ports { buttons_tri_i[6]}]
set_property IOSTANDARD LVCMOS33 [ get_ports { buttons_tri_i[6]}]

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