Laboratory Exercise #2
Using the Software Development Kit (SDK)

ECEN 449-511
Seth Pregler

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

Introduction
The purpose of this lab was to get familiar with the Vivado SDK and implement the MicroBlaze softcore processor and GPIO using the Vivado Block Design Builder.

Procedure
First, we created the MicroBlaze softcore processor system using the Vivado Block Design Builder. Next, we added GPIO capabilities to the processor using the IP libraries provided in Vivado. Lastly, we created the software to run on the MicroBlaze processor inorder to implement the appropriate functionality.

This lab was broken down into two parts; the first part of the lab guided us through the process of setting up the block design, exporting our hardware to the SDK using an HDL wrapper, implementing the counter using the provided C code, and running the binary on the FPGA. In the second part of the lab, we were tasked with implementing different functionality on the board; the subsequent list describes this portion of the lab in greater detail.

1) Add an 8-bit GPIO IP block to the system and connect the lower 4-bits to the switches and the upper 4-bits to the push buttons on the ZYBO board.
2) Create a source file to implement the following functionality:
    a) Program should keep track of a count value.
    b) When button 0 is held down, the cont should increment at 1 Hz
    c) When button 1 is held down, the count should decrement at 1 Hz.
    d) When button 2 is held down, the status of the switches should be displayed on the LEDs.
    e) When the button 3 is held down, the count value should be displayed on the LEDs
    f) The console should display the current action and LEDs value whenever there is an event.

Results
My implementation of the system consisted of two GPIO blocks; one 4-bit GIPO block was dedicated to outputting values to the LEDs while the other was an 8-bit GPIO block interconnecting the buttons and switches to the processor. As mentioned previously, the upper 4-bits are for the buttons while the lower 4-bits are for the switches. This is important to note because in the software implementation, I used bitwise AND to manipulate the "bitstream" into something that was easy to work with. You will see in *Appendix C*, where I provided my source code for the counter, I used two helpful variables: `val_h` to control the logic and `val_l` whenever outputting to the LEDs.

One last thing to note is the importance of reading the Xilinx header files. Before analyzing xparamater and xgpio, it was not clear how I was going to interact with the GPIO created in Vivado using the Block Designer. These files included necessary structs and defined functions and objects that made the process very easy.

Conclusion
After completing this lab, I feel that I got acquainted with the Vivado SDK, the Vivado Block Design Builder and a bit of the MicroBlaze processor, however I would like to get more familiar with it in the future. This lab was my first time implementing hardware in a software environment. After a bit of initial

struggle, I believe I learned a lot and enjoyed doing so. I can see this type of development having real implications in that it allows for wider accessibility to embedded development (not just RTL engineers).

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?

The clock rate in this lab was 100 MHz as opposed to 125 MHz in the previous lab. This translates to 1,000,000 clock cycles per second.

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

Volatile tells the compiler not to optimize the variable count. Because we want the count to depend on the processor which is out of the scope of the program, we use it to prevent any unwanted compiler optimizations and force it to do what we wrote.

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

Run indefinitely.

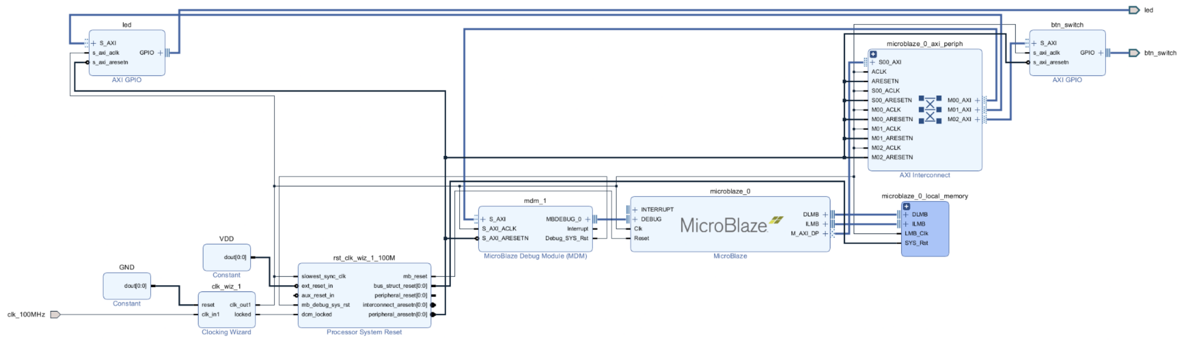
(d) Compare and contrast this lab with the previous lab. Which implementation do you feel is easier? What are the advantages and disadvantages associated with a purely software implementation such as this when compared to a purely hardware implementation such as the previous lab?

The ways in which we implemented logic in lab 1 and lab 2 were quite different. Using the SDK and C to program the FPGA, we had access to lots of useful libraries with prebuilt functionality, which I have not experienced using HDL. This allowed me to be more efficient and I don’t think I could have got as far using verilog alone. A possible disadvantage of the software implementation may relate to less flexibility in that you are limited to using xilinx provided libraries, however this is not substantiated in my brief experience using the SDK.

Appendices

List of Contents

Appendix A
Part II System Block Diagram



Appendix B
Part I Code

---

lab2a.c

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

#define GPIO_DEVICE_ID XPAR_LED_DEVICE_ID /* GPIO devices that LEDs are
connected */
#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, 0,  0x00);
    if (status != XST_SUCCESS) {
        xil_printf(" Initialization Failted");
    }

    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);
}
```

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

led.xdc

```
##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]}]

##Clock
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 clock_100MHz]
```

Appendix C
Part II Code

up_down_counter.c

```
#include <xparameters.h>
#include <xgpio.h>  // Header file for AXI GPIO
#include <xstatus.h>
#include <xil_printf.h>

// Define for peripherals for both LED and BTN/Switch GPIO blocks
according to xparameters.h
#define LED XPAR_LED_DEVICE_ID
#define BTN_SWITCH XPAR_BTN_SWITCH_DEVICE_ID
#define WAIT_VAL 10000000


int delay(void);

int main()
{
    int count, count_masked;
    int val, val_l, val_h;
    int status1, status2;

    XGpio led, btn_switch;  // XGpio structs... led(output),
btn_switch(input)

    // Initialize both GPIO blocks
    status1 = XGpio_Initialize(&led, LED);
    status2 = XGpio_Initialize(&btn_switch, BTN_SWITCH);

    // Set data direction for both GPIO blocks
    XGpio_SetDataDirection(&led, 1,  0x00); // Set data direction to
outpupt
    XGpio_SetDataDirection(&btn_switch, 1, 0xFF);   // Set data direction
to input

    // Verify the initialization of LEDs
    if (status1 != XST_SUCCESS) {
        xil_printf(" Initialization of LEDs Failed");
```

```
    }

    // Verify the initialization of buttons and switches
    if (status2 != XST_SUCCESS) {
        xil_printf(" Initialization of Buttons and Switches Failed");
    }

    count = 0;
    count_masked = 0;
    while (1)
    {
        // Read value of input GPIO block
        val = XGpio_DiscreteRead(&btn_switch, 1);
        val_l = val & 0x0F; // Grab lowest 4 bits for switches
        val_h = val & 0xF0; // Grab upper 4 bits for buttons

        if (val_h == 0x10) // if btn0 is pressed, increment count
        {
            count++;
            count_masked = count & 0xF;
            xil_printf("Value of count = 0x%x \n\r", count_masked);

        }

        else if (val_h == 0x20) // if btn1 is pressed, decrement count
        {
            count--;
            count_masked = count & 0xF;
            xil_printf("Value of count = 0x%x \n\r", count_masked);
        }

        else if (val_h == 0x40) // if btn3 is pressed, display status of
switches
        {
            XGpio_DiscreteWrite(&led, 1, val_l); // Display upper 4 bits
to LEDs
            xil_printf("Value of switches: 0x%x \n\r", val_l);

        }
```

```
        else if (val_h == 0x80)
        {
            count_masked = count & 0xF;
            XGpio_DiscreteWrite(&led, 1, count_masked);
        }
        delay();
    }
    return (0);
}

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

led.xdc

```
##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]}]

## btns_4bits_tri_i
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[4]}]; #IO_L12N_T1_MRCC_35 Sch=btn[0]
```

```
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[5]}]; #IO_L24N_T3_34 Sch=btn[1]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[6]}]; #IO_L10P_T1_AD11P_35 Sch=btn[2]
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[7]}]; #IO_L7P_T1_34 Sch=btn[3]

## DIP_Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[0]}]; #IO_L19N_T3_VREF_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[1]}]; #IO_L24P_T3_34 Sch=sw[1]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[2]}]; #IO_L4N_T0_34 Sch=sw[2]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports
{btn_switch_tri_i[3]}]; #IO_L9P_T1_DQS_34 Sch=sw[3]

##Clock
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 clock_100MHz]
```

Appendix D
Output of TCL Console

TCF Debug Virtual Terminal - MicroBlaze Debug Module at USER2

```
Value of count = 0x1
Value of count = 0x2
Value of count = 0x3
Value of count = 0x4
Value of count = 0x3
Value of switches: 0x9
Value of count = 0x2
Value of count = 0x1
Value of count = 0x0
Value of count = 0x1
Value of count = 0x2
Value of count = 0x3
Value of count = 0x4
Value of switches: 0x9
Value of switches: 0xD
Value of switches: 0xD
Value of switches: 0xF
Value of count = 0x3
Value of count = 0x2
Value of count = 0x1
Value of count = 0x0
Value of count = 0xF
Value of count = 0xE
Value of switches: 0xF
```

```
    section, .heap: 0x00001d70 - 0x0000256f
    section, .stack: 0x00002570 - 0x0000296f

  0%    0MB   0.0MB/s  ??:?? ETA
100%    0MB   0.2MB/s  00:00

Setting PC to Program Start Address 0x00000000
Successfully downloaded H:/Seth/ECEN_449-Microprocessor_Sys_Design/lab/lab2b/lab2.sdk/up_down_counter_sw/Debug/up_down_counter_sw.elf
Info: MicroBlaze #0 (target 6) Running

initializing
  0%    0MB   0.0MB/s  ??:?? ETA
 28%    1MB   2.2MB/s  ??:?? ETA
 50%    1MB   1.9MB/s  ??:?? ETA
 72%    2MB   1.8MB/s  ??:?? ETA
 92%    3MB   1.7MB/s  ??:?? ETA
100%    3MB   1.8MB/s  00:02
Info: tcfchan#11 closed
xsct% Info: MicroBlaze #0 (target 6) Stopped at 0x12f24 (Stop)

Downloading Program -- H:/Seth/ECEN_449-Microprocessor_Sys_Design/lab/lab2b/lab2.sdk/up_down_counter_sw/Debug/up_down_counter_sw.elf
    section, .vectors.reset: 0x00000000 - 0x00000007
    section, .vectors.sw_exception: 0x00000008 - 0x0000000f
    section, .vectors.interrupt: 0x00000010 - 0x00000017
    section, .vectors.hw_exception: 0x00000020 - 0x00000027
    section, .text: 0x00000050 - 0x00001783
    section, .init: 0x00001784 - 0x000017bf
    section, .fini: 0x000017c0 - 0x000017df
    section, .ctors: 0x000017e0 - 0x000017e7
    section, .dtors: 0x000017e8 - 0x000017ef
    section, .rodata: 0x000017f0 - 0x00001c7f
    section, .sdata2: 0x00001c80 - 0x00001c7f
    section, .data: 0x00001c80 - 0x00001da3
    section, .sdata: 0x00001da4 - 0x00001da7
    section, .sbss: 0x00001da8 - 0x00001da7
    section, .bss: 0x00001da8 - 0x00001dcf
    section, .heap: 0x00001dd0 - 0x000025cf
    section, .stack: 0x000025d0 - 0x000029cf

  0%    0MB   0.0MB/s  ??:?? ETA
100%    0MB   0.2MB/s  00:00

Setting PC to Program Start Address 0x00000000
Successfully downloaded H:/Seth/ECEN_449-Microprocessor_Sys_Design/lab/lab2b/lab2.sdk/up_down_counter_sw/Debug/up_down_counter_sw.elf
Info: MicroBlaze #0 (target 6) Running

xsct%
```