ECEN 749 Lab 2 Report

Tong Lu (UIN 621007982)

September 18, 2019

TEXAS A&M
UNIVERSITY®

Introduction

In Lab 2 we crearted a LED control project from a software based approach. The project has a similar functionality as Lab 1, but this time we created a soft-core microprocessor on Vivado, and then developped a C program using the Xilinx SDK on top of processor.

Procedure

Part A

1. Launch Vivado and create the MicroBlaze block design based on the lab manual.
2. Add constraints file into the source and map the I/O ports to LEDs and buttons.
3. Validate the block design, create the top level module for the design, generate bitstream and finally export the hardware platform.
4. Launch SDK and create a C program on top of the hardware platform.
5. Save the C application, connect the FPGA, program the FPGA, configure and run the C program on FPGA.

Part B

1. Launch Vivado and create the MicroBlaze block design similar to Part A. Then add dual channel 8-bit GPIO IP block, connect the lower 4-bit channel to switches, and connect the upper 4-bit channel to push buttons (Figure 1).

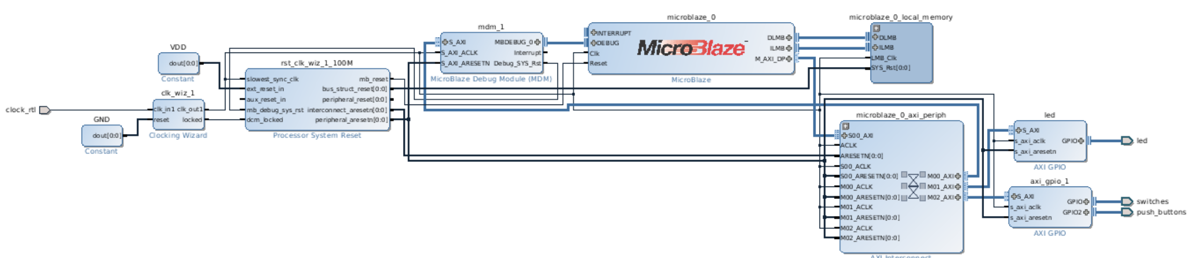

Figure 1. Block Design

2. Add constraints file similar to Part A, and also map the 8-bit GPIO to LEDs and buttons (See led_sw_counter.xdc in Appendix).
3. Synthesis the block design and export the hardware platform.
4. Launch SDK and create a C program on top of the hardware platform (See lab2b.c in Appendix). To make the LEDs display different things, a switch statement is implemented on button status varibles. When different button is pressed, the LEDs will display different informations, as well as the TCL console. Two flags varibles are used to record the rpevious status of button3 and button4, to pervent TCl terminal from printing out so many same messages when button3 or button4 is holded.
5. Save the C application, connect the FPGA, program the FPGA, configure and run the C program on FPGA.

Result

All the programs was finished and demostrated to TA. The programs are working well and meet all the requirement on lab manual (Figure 2).

```
COUNT = 0x0 (Display COUNT on LEDs);      LEDs = 0x0
switches = 0x1 (Display switches status); LEDs = 0x1
COUNT = 0xF (Decrement);                  LEDs = 0x0
COUNT = 0x0 (Increment);                  LEDs = 0x0
COUNT = 0x1 (Increment);                  LEDs = 0x0
switches = 0x0 (Display switches status); LEDs = 0x0
COUNT = 0x1 (Display COUNT on LEDs);      LEDs = 0x1
COUNT = 0x2 (Increment);                  LEDs = 0x0
COUNT = 0x3 (Increment);                  LEDs = 0x0
COUNT = 0x4 (Increment);                  LEDs = 0x0
COUNT = 0x5 (Increment);                  LEDs = 0x0
COUNT = 0x6 (Increment);                  LEDs = 0x0
COUNT = 0x5 (Decrement);                  LEDs = 0x0
COUNT = 0x4 (Decrement);                  LEDs = 0x0
COUNT = 0x4 (Display COUNT on LEDs);      LEDs = 0x4
switches = 0x4 (Display switches status); LEDs = 0x4
```

Figure 2. TCL Console Printout

Conclusion

In this lab I learned the to create the MicroBlaze soft core microprocessor and build C program on top to implemented some tasks. This lab is a good introduction for me and it will help me a lot in the future projects.

Answer to Questions

a Compared to the count value I used as a delay in the previous lab, the count value in this lab is very small. This is because we are running the C program on top of the MicroBlaze microprocessor core, instead of running all the logic directly on hardware in Lab 1. During the run time, the machine code compiled from C program will takes much longer time to perform same tasks on FPGA, due to the extra hardware and software configuration. In this lab, the on-board clock runs at 125MHz, and WAIT_VAL is set to 10M to make the one-second delay. Therefore, it needs $125 \div 10 \approx 12$ cycles to execute one iteration of the delay for-loop.

b Using volatile on varible, compiler will know that the variable might be accessed or written by other program or hardware. Therefore, the complier won't do optimazion to this varible.

c while(1) tell the processor to run the processor to keep running the commands and never stop.

d The implementation in lab 2 is easier. This is because the C complier and the MicroBlaze soft processor did a lot of job for us and so we just need to focus on the Algorithm. Compared to the pure hardware implementation, software implementation is easy to develop and maintain. The disadvantage however, is that the result won't be as optimazed performance wise as in the pure hardware implementations. Also, it will take more space and memory, since the FPGA needs to store the whole microtroller hardware design as well as the software program, whereas the pure hardware design only require a small piece of memory for the hardware design.

Appendix

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

/* Definitions */
#define GPIO_DEVICE_ID_LED XPAR_LED_DEVICE_ID   /* GPIO device that LEDs are connected to */
#define GPIO_DEVICE_ID_SW_BTN XPAR_AXI_GPIO_1_DEVICE_ID   /* GPIO device that LEDs are connected to */

#define WAIT_VAL 0x10000000

int delay(void);

int main() {

    int count;
    int count_masked;
    XGpio leds;
    XGpio sws_btns; // switches and buttons
    int status;
    short sw_status, btn_status; // Varibles that store the status of switch buttons (sw_status) and push buttons (btn_status
    short btn3_flg, btn4_flg; // Flag varibles to store the status of button3 and button4. This flag is used to prevent the countinuous print out of current action on TCL console when button3 or button4 is holding down
    // Initalize LEDs
    status = XGpio_Initialize(&leds, GPIO_DEVICE_ID_LED);
    XGpio_SetDataDirection(&leds, 1, 0x00);
    if(status != XST_SUCCESS) { // Error handler
        xil_printf("Initialization failed");
    }
    // Initalize switches and push buttons on the dual-channel 8-bit GPIO
    status = XGpio_Initialize(&sws_btns, GPIO_DEVICE_ID_SW_BTN);
    XGpio_SetDataDirection(&sws_btns, 1, 0x01); // Switches: GPIO channel 1, set as input
    XGpio_SetDataDirection(&sws_btns, 2, 0x01); // Push buttons: GPIO channel 1, set as input
    if(status != XST_SUCCESS) { // Error handler
        xil_printf("Initialization failed");
    }
    // Initalize local varibles
    count = 0;
    btn3_flg = 0;
    btn4_flg = 0;

    while(1) {
        sw_status = XGpio_DiscreteRead(&sws_btns, 1); // read switches status (GPIO channel 1)
        btn_status = XGpio_DiscreteRead(&sws_btns, 2); // read buttons status (GPIO channel 2)
        switch(btn_status) { // Using switch cases for different button status
        case 0b0001: // Button 1 is pressed
            count++; // COUNT increment
```

```c
        count_masked = count & 0xF; // Since we only have 4 LEDS, we just read the lower 4 bits of the counter here
        xil_printf("COUNT = 0x%x (Increment);                  LEDs = 0x0\n\r", count_masked); // Print out LED value as well as the current action on TCL console
        delay();
        if (btn3_flg == 1) btn3_flg = 0; // Since button1 was pressed in this loop cycle, set the button3 flag to 0
        if (btn4_flg == 1) btn4_flg = 0; // Since button1 was pressed in this loop cycle, set the button4 flag to 0
        break;
    case 0b0010: // Button 2 is pressed
        count--; // COUNT decrement
        count_masked = count & 0xF; // Since we only have 4 LEDS, we just read the lower 4 bits of the counter here
        xil_printf("COUNT = 0x%x (Decrement);                  LEDs = 0x0\n\r", count_masked); // Print out LED value as well as the current action on TCL console
        delay();
        if (btn3_flg == 1) btn3_flg = 0; // Since button2 was pressed in this loop cycle, set the button3 flag to 0
        if (btn4_flg == 1) btn4_flg = 0; // Since button2 was pressed in this loop cycle, set the button4 flag to 0
        break;
    case 0b0100: // Button 3 is pressed
        if (btn3_flg == 0) { // If button3 wasn't pressed in the previous loop cycle
            xil_printf("switches = 0x%x (Display switches status); LEDs = 0x%x\n\r", sw_status, sw_status); // Display status of switches
            XGpio_DiscreteWrite(&leds, 1, sw_status); // Write the switch status to LEDs
            btn3_flg = 1; // Since button3 was pressed in this loop cycle, set the button3 flag to 1
        }
        else { // If button3 was pressed in the previous loop cycle
            XGpio_DiscreteWrite(&leds, 1, sw_status); // Then only display switches statys on LEDs, don't print the same message on TCL console
        }
        if (btn4_flg == 1) btn4_flg = 0; // Since button3 was pressed in this loop cycle, set the button4 flag to 0
        break;
    case 0b1000: // Button 4 is pressed
        if (btn4_flg == 0) { // If button4 was not pressed in the previous loop cycle
            count_masked = count & 0xF; // Get the lower 4 bit of the count varible
            xil_printf("COUNT = 0x%x (Display COUNT on LEDs);      LEDs = 0x%x\n\r", count_masked, count_masked); // Print out current actuon and LED values on TCL console
            XGpio_DiscreteWrite(&leds, 1, count_masked); // Display COUNT on LEDs
            btn4_flg = 1; // Since button3 was pressed in this loop cycle, set the button4 flag to 1
        }
        else { // If button4 was pressed in the previous loop cycle
            XGpio_DiscreteWrite(&leds, 1, count_masked);// Then only display COUNT on LEDs, don't print the same message on TCL console
        }
        if (btn3_flg == 1) btn3_flg = 0; // Since button4 was pressed in this loop cycle, set the button3 flag to 0
        break;
    default: // In all other cases
```

```c
            XGpio_DiscreteWrite(&leds, 1, 0); // Turn off LEDs
            if (btn3_flg == 1) btn3_flg = 0; // Set button3 flag to 0
            if (btn4_flg == 1) btn4_flg = 0; // Set button4 flag to 0
            break;
        }
    }
    return (0);
}

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

src/lab2b.c

```
## 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
    ]

## 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]}]

## Switches: connect lower 4 bits of GPIO to switches
set_property PACKAGE_PIN G15 [get_ports {switches_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_tri_i[0]}]

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

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

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

## Push Buttons: connect upper 4 bits of GPIO to push buttons
set_property PACKAGE_PIN R18 [get_ports push_buttons_tri_i[0]]
set_property IOSTANDARD LVCMOS33 [get_ports push_buttons_tri_i[0]]

set_property PACKAGE_PIN P16 [get_ports push_buttons_tri_i[1]]
set_property IOSTANDARD LVCMOS33 [get_ports push_buttons_tri_i[1]]

set_property PACKAGE_PIN V16 [get_ports push_buttons_tri_i[2]]
set_property IOSTANDARD LVCMOS33 [get_ports push_buttons_tri_i[2]]
```

```

set_property PACKAGE_PIN Y16 [get_ports push_buttons_tri_i[3]]
set_property IOSTANDARD LVCMOS33 [get_ports push_buttons_tri_i[3]]
```

src/led_sw_counter.xdc