

ECEN 449 - Lab 2 Report

Lab Number: 2

Lab Title: Using the Software Development Kit (SDK)

Section Number: 511

Student's Name: Kris Gavvala

Student's UIN: 929000158

Date: 2/19/2025

TA: Gautham Nemani

Introduction

This lab's goal was to learn how to develop a software-based solution for led controls as opposed to the hardware designs in lab 1. Vivado's Block Design Builder was used to implement a MicroBlaze soft processor and extend its functionality by adding GPIO capabilities through Xilinx IP blocks. The MicroBlaze soft processor was programmed with C to enable LED control and interaction with DIP switches and push buttons. The processor was then uploaded to the ZYBO- Z7-10 to test its functionality with the board's LEDs, switches, and buttons.

Procedure

- Create a new RTL project in Vivado without adding source files

Block Design:

- In Vivado, click 'Create a block design' to open the block design editor, add the MicroBlaze processor then run block automation.
- Add constraints in Microblaze to enable Local Memory, Debug Module, and clock
- Add AXI GPIO ports and add appropriate constraints.
- Run connection automation and regenerate the layout.

Mapping ports to ZYBO FPGA:

- In Vivado, add constraints by creating an XDC file
- Map the clock and gpio pins on the fpga to the ports on the Microblaze processor system
- Use tri_i or tri_o for ports based on inputs or outputs.

Bitstream Generation:

- Validate the block design.
- Create an HDL wrapper.
- Generate bitstream and Export hardware (include bitstream) as a .xsa file.

Vitis IDE:

- create a new application project in Vitis IDE from the .xsa file imported.
- Add the C code to program the soft processor
- Build the hardware and software projects and program the FPGA
- Attach the Vitis console and run the application to observe the program behavior.

Results

In the first part, the processor was programmed such that it incremented a counter every second and displayed the binary value on the leds of the fpga. Since 1111 was the highest value displayed by the leds, the leds would cycle back to 0000 every 15 seconds and resume counting. For the second part, the functionality was implemented such that button 0 and 1 could increase the count every second when held, button 2 could display the count on the leds, and button 4 could display the count values represented by the fpga switches.

Conclusion

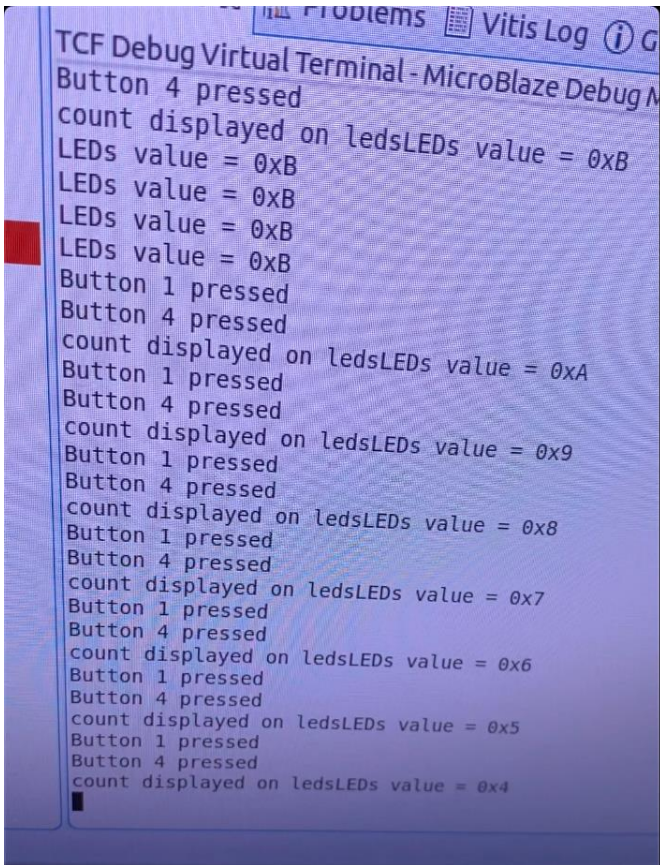
The first part of the lab was to learn how to implement a soft processor system using microblaze, and to program and run it on the Zybo Z7-10 board. The second part focused more on C programming, providing some experience on how to program a processor and communicate with the gpios using libraries like Xgpio. I also learned some strategies for reading and writing data such as using masks to get the desired bit range of a value.

Post-lab Deliverables

1. [4 points.] check
2. [5 points.] check – code in appendix
3. [5 points.] check

4. [2 points.] The output of the TCL console from the part 2 demo.

Figure 1: part2 - output from TCL console



```
TCF Debug Virtual Terminal - MicroBlaze Debug M
Button 4 pressed
count displayed on ledsLEDs value = 0xB
LEDs value = 0xB
LEDs value = 0xB
LEDs value = 0xB
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0xA
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x9
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x8
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x7
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x6
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x5
Button 1 pressed
Button 4 pressed
count displayed on ledsLEDs value = 0x4
```

5. [4 points.]

(a) how many clock cycles are required to execute one iteration of the delay for-loop? If so, how many?

Answer: 3 cycles. For one iteration of the while loop in the delay() function there are three instructions that get executed : cmp, addi, blt. Each of these instructions takes one cycle to complete. So there are **three cycles** used for one iteration of the while loop.

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

Answer: It prevents the compiler from optimizing the code that uses the delay_count variable. It ensures that the variable is re-read from memory every time it is accessed.

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

Answer: it means while(true), it just runs an infinite loop.

(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?

Answer:

This lab dealt with implementing logic through software, and the other dealt with implementing it through hardware description language. In the last lab your Verilog code had to synthesize to hardware so delays could not be used, and you had to synchronize with clock edge triggers. In this lab all logic is implemented through C, so you can use delays in the code. This lab was easier.

Software implementations are easier to implement and offer more flexibility. The logic used in C programming is easy to understand and is very quick to implement and test. Hardware implementations can be more complex; synchronous logic has more parts to it and requires knowledge of how Verilog synthesizes. With Verilog, you are limited based on hardware capabilities, and for complex logic it takes more time to implement with Verilog than with C. Hardware logic is better for performance and resource efficiency since you have precise control of synchronous logic. A counter in Verilog takes one clock cycle to increment, but the same counter implemented in C takes multiple cycles. Software implementations are inefficient and unlike hardware, cannot perform parallelism to improve performance.

Appendix

Figure 2: led.xdc

```
## CLOCK_RTL

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]]

#switches

set_property PACKAGE_PIN G15 [get_ports inputs_tri_i[0]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[0]]

set_property PACKAGE_PIN P15 [get_ports inputs_tri_i[1]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[1]]

set_property PACKAGE_PIN W13 [get_ports inputs_tri_i[2]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[2]]

set_property PACKAGE_PIN T16 [get_ports inputs_tri_i[3]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[3]]

#buttons

set_property PACKAGE_PIN K18 [ get_ports inputs_tri_i[4]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[4]]

set_property PACKAGE_PIN P16 [ get_ports inputs_tri_i[5]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[5]]

set_property PACKAGE_PIN K19 [ get_ports inputs_tri_i[6]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[6]]

set_property PACKAGE_PIN Y16 [ get_ports inputs_tri_i[7]]
set_property IOSTANDARD LVCMOS33 [ get_ports inputs_tri_i[7]]
```

Figure 3: lab2b.c

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

/*
Definitions
*/

#define GPIO_LEDS_ID XPAR_LED_DEVICE_ID
#define GPIO_INPUTS_ID XPAR_INPUTS_DEVICE_ID
/*
GPIO device that LEDs are connected to
*/

#define WAIT_VAL 100000000

int delay(void);

int main()
{
    int count;
    int count_masked;
    XGpio leds;
    XGpio inputs;
    int inputs;
    int statusLeds;
    int statusInputs;

    statusLeds = XGpio_Initialize(&leds, GPIO_LEDS_ID);
    XGpio_SetDataDirection(&leds, 1, 0x00); //initialize leds to output

    statusInputs = XGpio_Initialize(&inputs, GPIO_INPUTS_ID);
    XGpio_SetDataDirection(&inputs, 1, 0xFF); //initialize inputs

    if (statusLeds != XST_SUCCESS)
    {
        xil_printf("leds Initialization failed");
    }

    if (statusInputs != XST_SUCCESS)
    {
        xil_printf("inputs Initialization failed");
    }
}
```

Figure 4: lab2b.c continued

```
count = 0;

while (1)
{
    // read from inputs

    inputs = XGpio_DiscreteRead(&inputs, 1);

    // check if button0 is pressed, it is the 5th bit.

    if (inputs & 0x10)
    {
        count++;
        xil_printf("Button 0 pressed\n\r");
    }

    // if button0 is on increment count by 1

    if (inputs & 0x20)
    {
        count--;
        xil_printf("Button 1 pressed\n\r");
    }

    // if button1 is pressed decrement count by 1

    if (inputs & 0x40)
    {
        xil_printf("Button 2 pressed\n\r");
        int switches = inputs & 0x0F; //use mask to get first 4 bits
        xil_printf("Switches value = 0x%x\n\r", switches);
    }

    // if button2 is pressed display the status of the switches

    if (inputs & 0x80)
    {
        xil_printf("Button 4 pressed\n\r");
        count_masked = count & 0xF; //use mask to get last 4 bits
        XGpio_DiscreteWrite(&leds, 1, count_masked);
        xil_printf("count displayed on leds 0x%x\n\r", switches);
    }

    // if button4 is pressed display count on the leds

    // console should display the current action and leds value

    xil_printf("LEDs value = 0x%x\n\r", count_masked);

    delay();
}

return 0;
}

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

    return 0;
}
```