Laboratory Exercise #2

Using the Software Development kit (SDK)

Name: Daniel Horan
UIN: 527005307

ECEN 449-504

I. Introduction

This lab focuses on leveraging the Vivado Block Design Builder to establish a MicroBlaze processor system. By integrating General Purpose Input/Output (GPIO) capabilities from Xilinx's Intellectual Property (IP) hardware blocks, we enhance the microprocessor's functionality. The core objective is to develop software in the C programming language, tailored to run on the MicroBlaze processor, enabling precise LED control. This exercise highlights the seamless integration of software and hardware in FPGA design.

II. Procedure

The lab was organized into clear steps, starting with setting up the Vivado environment. We began by creating a directory and launching Vivado. A new RTL project was initiated, with a focus on integrating the Xilinx Microblaze Processor.

Next, we worked with the IP Integrator to create a block design. This involved adding the MicroBlaze processor and configuring it with specific settings. We also integrated the Clocking Wizard and added General Purpose IO (GPIO) blocks to interact with the ZYBO Z7-10 board's LEDs, switches, and buttons.

We then mapped the IO ports to the LEDs and buttons. This required creating a constraints file and connecting the ports to the ZYBO Z7-10 board. After setting up the necessary configurations, we generated the bitstream and transitioned to the Vitis IDE.

In Vitis, we developed a software application. This process included creating an application project and writing a specific C code. After building the software, we programmed the FPGA, resulting in a sequence display on the LEDs.

The final step introduced an 8-bit GPIO IP block. We connected it to the switches and buttons on the ZYBO Z7-10 board. The software tracked a COUNT value, with push buttons assigned specific functions.

III. Results

During this lab, the primary focus was on leveraging Vivado to develop a software-based LED control solution. The initial stages, which involved launching Vivado and crafting a block design, proceeded without major hitches. However, a challenge arose when attempting to program the board with Xilinx, specifically concerning the elf file directory. After some troubleshooting, this was resolved, and the LEDs accurately reflected the count.

A significant challenge was encountered during the integration of the 8-bit GPIO. Initially, the buttons did not respond as anticipated. Two potential reasons were identified for this behavior:

1. Possible incorrect integration of the 8-bit GPIO in the design.
2. Potential missteps in defining the switches and buttons in the C code, especially given the lack of guidance from the xparameters.h file, necessitating reliance on external online sources.

To further diagnose the issue, I have done more tests on the software part. After running the C code for the 8-bit GPIO, it was determined that the code was functioning correctly. This revelation suggests that the challenges faced were likely due to either mistakes in the design phase or issues with the synthesis of my files.

In summary, while many lab objectives were met and the C code was validated to be correct, the challenges with the 8-bit GPIO and button operations highlight the intricacies of hardware-software integration and the importance of thorough testing and collaboration. Future endeavors will involve revisiting the design and synthesis stages to pinpoint and rectify the root cause of the observed issues.

IV. Conclusion

In this lab, we explored FPGA design using the ZYBO Z7-10 board and Vitis IDE. Successfully programming the FPGA with Microblaze and implementing the LED counter demonstrated the practical application of our theoretical knowledge. However, challenges with the 8-bit GPIO integration highlighted the importance of precision in both hardware setup and software coding. This experience emphasized the need for careful initialization and GPIO management. Overall, the lab provided valuable insights into the nuances of microprocessor design.

V. Questions

a) In Lab 1, we used a hardware-based delay with a count of 25,000,000, achieved through a clock divider in the FPGA design. This method directly leveraged the FPGA's clock cycles. In contrast, Lab 2's software-based delay utilized a count of 10,000,000, executed on the processor. Each iteration of this software loop consumes about 6 clock cycles, a value influenced by the processor's architecture and the specific compiler optimizations applied.

b) Since the count variable may change at any time, volatile is used for the delay_count variable to prevent the compiler from optimizing away the delay loop. Without volatile, the compiler might see the loop as redundant and remove it, eliminating the intended delay. By using it, we ensure the loop remains intact during compilation, preserving the delay functionality.

c) The while(1) expression in the code creates an infinite loop. Within this loop, the LEDs are continuously updated based on the count variable. The loop will run indefinitely, repeatedly updating the LED values, printing the current LED value to the console, introducing a delay, and incrementing the count. This ensures that the LED behavior persists for the duration the program runs.

d) In comparing the two labs, the purely hardware implementation from the previous lab felt more direct and presented fewer challenges than the current software-based lab. One notable advantage of the software approach is the utilization of a familiar programming language, which can streamline the development process for those well-versed in it. However, a significant drawback is its less intuitive nature, especially evident when I encountered difficulties with the 8-bit GPIO implementation. While hardware designs offer a tangible and often clearer representation of logic, software implementations might introduce complexities due to the abstraction layers involved.

VI. Appendix

Lab2a.c:

```
#include <xparameters.h> //Provides definitions for the Xilinx parameters, such as device IDs and base addresses
#include <xgpio.h> //Xilinx GPIO (General Purpose Input/Output) driver, which provides functions to control GPIOs
#include <xstatus.h> //Provides status codes (like XST_SUCCESS) for Xilinx drivers and utilities
```

```
#include <xil_printf.h> //Xilinx printf function, which allows for formatted output to the console

/* Definitions */
#define GPIO_DEVICE_ID XPAR_LED_DEVICE_ID //Defines the GPIO device ID for the LEDs
#define WAIT_VAL 10000000 //Defines a constant for the delay function to determine how long the
delay should be

int delay(void); //Delay function

int main() {
   int count;    //Integer variable to keep track of the count
   int count_masked;    //Variable to store the masked value of the count
   XGpio leds;    //GPIO instance for the LEDs
   int status;    //Variable to store the status of GPIO initialization

   status = XGpio_Initialize(&leds, GPIO_DEVICE_ID);    //Initializes the GPIO for the LEDs and
store the status
   XGpio_SetDataDirection(&leds, 1, 0x00);    //Sets the data direction for the GPIO (in this case, set
as output)

   if (status != XST_SUCCESS) {    //Checks if the GPIO initialization was successful
      xil_printf("Initialization failed");       //Prints an error message if initialization failed
   }

   count = 0;
   while (1) {    // Sets an infinite loop to continuously update the LEDs
      count_masked = count & 0xF;       //Masks the count value to get the lower 4 bits
      XGpio_DiscreteWrite(&leds, 1, count_masked);       //Writes the masked count value to the
LEDs
      xil_printf("Value of LEDs = 0x%x\n\r", count_masked);       //Prints the current value of the
LEDs to the console
      delay(); //Calls the delay function; delay between LED updates
      count++;
   }

   return (0);
}

int delay(void) { //The delay function
   volatile int delay_count = 0;    //Declares a volatile integer variable for the delay count (volatile
ensures the compiler doesn't optimize it away)
   while (delay_count < WAIT_VAL) {   //Loops until the delay count reaches the defined
WAIT_VAL
      delay_count++;
   }
   return (0);
}
```

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]}]
```

lab2b.c:

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

#define GPIO_DEVICE_ID_LEDS XPAR_LED_DEVICE_ID //Defines the device ID for the LEDs
using the parameter from xparameters.h
#define GPIO_DEVICE_ID_SWB XPAR_GPIO_0_DEVICE_ID //Defines the device ID for the
switches and buttons
#define CLOCK_DELAY 10000000 //Defines a constant for the delay duration

int delay() { //Delay function definition
   int delay_counter = 0; // Counter for the delay

   while (delay_counter < CLOCK_DELAY) { //Increments the counter until it reaches the defined
delay duration
      delay_counter++;
   }
   return 0;
}

int main() {
   int COUNT = 0;
   int COUNT_MASKED = 0;
   XGpio leds, swb; //Declares variables for the LED and switch/button GPIO instances
   int init_status; //Declares the initialization status
```

```
    init_status = XGpio_Initialize(&leds, GPIO_DEVICE_ID_LEDS); //Initializes the LED GPIO
instance

    if (init_status != XST_SUCCESS) { //Checks if the LED initialization was successful
        xil_printf("LED Initialization failed\n");
        return XST_FAILURE;
    }

    XGpio_SetDataDirection(&leds, 1, 0); //Sets the data direction for the LED GPIO instance (output)
    init_status = XGpio_Initialize(&swb, GPIO_DEVICE_ID_SWB); //Initializes the switch/button
GPIO instance

    if (init_status != XST_SUCCESS) { //Checks if the switch/button initialization was successful
        xil_printf("Switch/Button Initialization failed\n");
        return XST_FAILURE;
    }

    XGpio_SetDataDirection(&swb, 1, 0xFFFFFFFF); //Sets the data direction for the switch/button
GPIO instance (input)

    int previous_swb_value = 0; //Detects button state changes

    while (1) { //Continuously checks the status of switches and buttons; infinite loop
        int swb_value = XGpio_DiscreteRead(&swb, 1); //Reads the value from the switch/button GPIO
instance
        int push_buttons = (swb_value & 0xF0) >> 4; //Extracts the push button values from the higher 4
bits
        int switches = swb_value & 0x0F; //Extracts the switch values from the lower 4 bits

        if (push_buttons != (previous_swb_value & 0xF0)) { //Checks for button state changes
            delay(); //Debounce delay

            if (push_buttons & 0x1) { //Checks if the first push button is pressed
                COUNT++;
                COUNT_MASKED = COUNT & 0xF;
                xil_printf("Incrementing COUNT. Current COUNT: %d\n", COUNT);
            }
            else if (push_buttons & 0x2) { //Checks if the second push button is pressed
                COUNT--;
                COUNT_MASKED = COUNT & 0xF;
                xil_printf("Decrementing COUNT. Current COUNT: %d\n", COUNT);
            }
            else if (push_buttons & 0x4) {     //Checks if the third push button is pressed
                xil_printf("Switch status: %d\n", switches);
            }
            else if (push_buttons & 0x8) {      //Checks if the fourth push button is pressed
                XGpio_DiscreteWrite(&leds, 1, COUNT_MASKED);
                xil_printf("Displaying COUNT on LEDs. COUNT: %d\n", COUNT);
```

```
            }
        }

        previous_swb_value = swb_value; //Updates the previous button state
    }
    return (0);
}
```

lab2b.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; Programming LEDs
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]}]

#swb_tri_i; Programming Buttons/Switches
##Buttons
set_property PACKAGE_PIN K18 [get_ports {swb_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[0]}]

set_property PACKAGE_PIN P16 [get_ports {swb_tri_i[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[1]}]

set_property PACKAGE_PIN K19 [get_ports {swb_tri_i[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[2]}]

set_property PACKAGE_PIN Y16 [get_ports {swb_tri_i[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[3]}]
##Switches
set_property PACKAGE_PIN G15 [get_ports {swb_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[4]}]

set_property PACKAGE_PIN P15 [get_ports {swb_tri_i[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[5]}]

set_property PACKAGE_PIN W13 [get_ports {swb_tri_i[6]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[6]}]

set_property PACKAGE_PIN T16 [get_ports {swb_tri_i[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {swb_tri_i[7]}]
```