Mauro Lopez

# LAB 2 Report

ECEN-449

Section 502

September 20, 2018

## Introduction:

In this lab we are going to learn how to implement a different method to program our FPGA board. We will be using the same concept of a counter, from the previous lab. We will be using a MicroBlaze processor and adding GPIOs, and finally we will use C language to program the GPIOs and processor to manage the logic of the FPGA board.

## Procedure:

We start the lab by first creating a new project in Vivado. This first step is the same for lab1.

Once we are in our main screen of Vivado, we can see a “IP integrator section” we will be using this to create our MicroBlaze diagram. We begin by creating a new block design. We are given the specs that we will be using for this block, use those to create the diagram.

We then change our clock wizard block and set it to use “single ended clock capable pin”. After we have completed that, we right click run connection automation, and select all.

We then add our AXI GPIO block that we will map to our LEDs. Since these will all be output and only 4 are going to be use. We set up our block names to led, and our width to 4 and all outputs.

We run the automation and we can regenerate our layout to clean things up.

Finally, we will be changing our input reset to be set to constants for this lab.

We add two constants block and replace our two reset inputs. We are not expected to map these two buttons for the lab, so we use constants instead.

Once we have completed the steps above, we need to check and see if we have everything connected properly, by clicking the validate design.

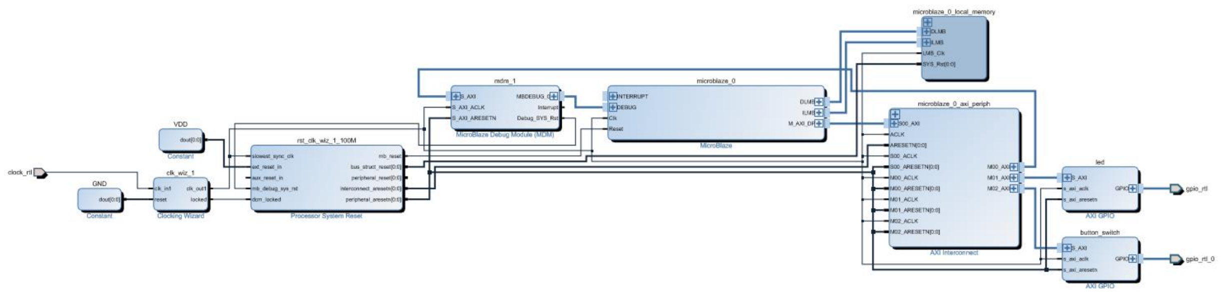
We then go to project manager and we create HDL wrapper. I prefer to do this, before creating the .xdc file, since this will give you the name of your input and output variables.

We are given the following code for our .xdc file:

```
#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 {gpio_rtl_tri_o[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[0]}]

set_property PACKAGE_PIN M15 [ get_ports {gpio_rtl_tri_o[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[1]}]

set_property PACKAGE_PIN G14 [ get_ports {gpio_rtl_tri_o[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[2]}]

set_property PACKAGE_PIN D18 [ get_ports {gpio_rtl_tri_o[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[3]}]

```

*note : my variables are different from lab manual.

Our diagram should look similar to this at the end.



Once we have our .xdc file and our HDL wrapper we can generate our bit stream.

Now we can export our bit stream and launch SDK. With SDK open, we create a new empty project. We go into our documents sources and add a new file. call this file “lab2a.c”. You can edit this file directly in this IDE.

Base on the code we were given we get this :

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

#define GPIO_DEVIDE_ID XPAR_LED_DEVICE_ID //found in xparameter.h
#define WAIT_VAL 10000000 // count goal

int delay(void );
int main(){
    //declaring our variables
    int count;
    int count_masked;
    XGpio leds;

    int status;
    //initializing our GPIO to leds.
    status = XGpio_Initialize(&leds, GPIO_DEVIDE_ID);

    XGpio_SetDataDirection(&leds,1,0x00);// function to update LEDs.
    if (status != XST_SUCCESS) {
        xil_printf( "Initialization failed " ) ;
    }
    count =0;
    while(1){
        count_masked = count & 0xF;// update Leds
        XGpio_DiscreteWrite(&leds,1,count_masked) ; // update GPIO
        xil_printf("Value of LEDS = 0x%x\n\r", count_masked);
        delay();
        count++;
    }
    return (0);
}


int delay(void){ // function definition for delay.
    volatile int delay_count=0;
    while(delay_count<WAIT_VAL){
        delay_count++;
    }
    return (0);
}
```

This code will keep track of the counter and display in both the FPGA board and the console.

We then program our FPGA board using the Xilinx tools.

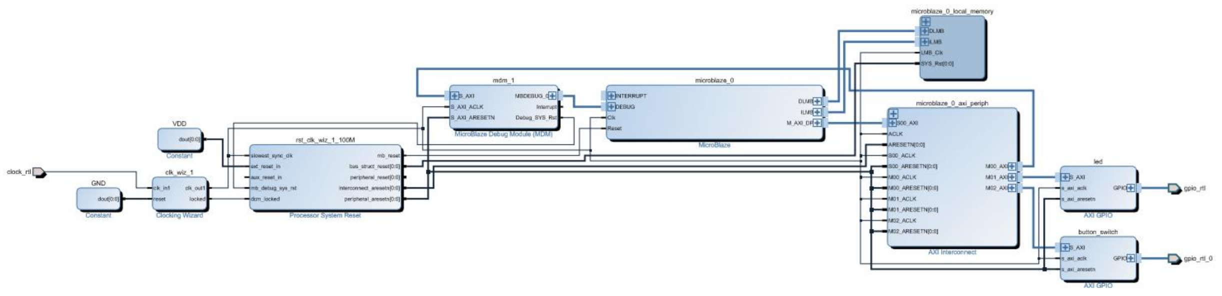
We also need to update our debug configuration to keep the debugger connected to our c file.

## Part 2:

Now that we have an idea of what each component does. We can begin to work on part 2 and use what we learn to create a new counter. Except this time, we will be adding switches and buttons to our design.

Using our previous diagram for our MicroBlaze processor we can add another AXI GPIO, this time instead of all being outputs they will be outputs. The width will be of 8, since we are using both the 4 switches and the 4 buttons. We then run the connection automation, and regenerate our layout to cleans things up. Now we have to generate a new HDL wallpaper, and create an updated .xdc file.

We should end up with something similar to this :



For our .xdc file:

```
#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 {gpio_rtl_tri_o[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[0]}]

set_property PACKAGE_PIN M15 [ get_ports {gpio_rtl_tri_o[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[1]}]

set_property PACKAGE_PIN G14 [ get_ports {gpio_rtl_tri_o[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[2]}]

set_property PACKAGE_PIN D18 [ get_ports {gpio_rtl_tri_o[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_tri_o[3]}]

#buttons

set_property PACKAGE_PIN Y16 [ get_ports {gpio_rtl_0_tri_i[7]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[7]}]

set_property PACKAGE_PIN V16 [ get_ports {gpio_rtl_0_tri_i[6]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[6]}]

set_property PACKAGE_PIN P16 [ get_ports {gpio_rtl_0_tri_i[5]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[5]}]

set_property PACKAGE_PIN R18 [ get_ports {gpio_rtl_0_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[4]}]


#switches
set_property PACKAGE_PIN T16 [ get_ports {gpio_rtl_0_tri_i[3]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[3]}]

set_property PACKAGE_PIN W13 [ get_ports {gpio_rtl_0_tri_i[2]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[2]}]

set_property PACKAGE_PIN P15 [ get_ports {gpio_rtl_0_tri_i[1]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[1]}]

set_property PACKAGE_PIN G15 [ get_ports {gpio_rtl_0_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [ get_ports {gpio_rtl_0_tri_i[0]}]
```

and for our C file:

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


#define GPIO_DEVIDE_ID XPAR_LED_DEVICE_ID // can be found in the parameter file
#define BUTTON_DEVICE_ID XPAR_BUTTON_SWITCH_DEVICE_ID //also found in parameter file.

#define WAIT_VAL 10000000 // numerator use to count.

int delay(void );
int main(){
    int count; // keep track of our count
    int count_masked; // adjusted count to LEDs
    int switches;
    XGpio leds; //declare our two XGpios variables.
    XGpio buttonSwitch;
    int commands;

    int status;
    int value;
    int buttons;

    // initialize both of our variables.
    XGpio_Initialize(&buttonSwitch, BUTTON_DEVICE_ID);
    status = XGpio_Initialize(&leds, GPIO_DEVIDE_ID);

    //adjust our functions to manipulate our XGPIO variables.
    XGpio_SetDataDirection(&leds,1,0x00);
    XGpio_SetDataDirection(&buttonSwitch,1,0x00);

    // test to see if things run.
```

```c
    if (status != XST_SUCCESS) {
        xil_printf( "Initialization failed " ) ;
    }

    count =0;
    /// this next looop will keep executing and wont stop until user stops
    // the program.
    while(1){
        count_masked = count & 0xF;

        value=XGpio_DiscreteRead(&buttonSwitch,1); //update value of button and switch

        switches= value & 0x0F; // get a masked switches
        buttons= (value & 0xF0); // get a masked buttons.

        if(buttons==0x10){ //button 0
            commands=1;
            count++;
        }
        else if(buttons==0x20){ // button 1
            commands =2;
            count--;
        }
        else if(buttons ==0x40){ // button 2
            commands =4;
             xil_printf("switch value %x \n",switches);
        }
        else if(buttons == 0x80){ // button 3
            commands = 8;
            XGpio_DiscreteWrite(&leds,1,count_masked) ;
        }
        else{
            commands =0;
        }
        // print the status of each component.
        xil_printf("Value of LEDS = 0x%x\n\r", count_masked);

        xil_printf("buttons value %x \n\n",commands);
        delay();
        XGpio_DiscreteWrite(&leds,1,0);




    }
    return (0);
}


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

Once you have both your diagram and .xdc files complete. Generate you new bit stream and exported to be use by the SDK program. Create a new project, update your debugger, program you FPGA board and run the code.

**Results:**

**Part 1:**

Following the instructions on the lab manual, we should be getting a counter that counts up at 1Hz, with no user interactions. Running my code I was able to run the counter, to show up in both the FPGA board and console. The counter however was using hexadecimals on console than standard decimal numbers.

**Part 2:**

Updating part 1 to meet part 2 expectations required to repeat all of the processes and allow me to become more familiar. The final result for part 2, was a success and both the console and FPGA board agree on the output. Each button performed the correct action.

**Conclusion:**

In this lab we got to experience working on a new method to program an FPGA board. Although its similar our usual module and Verilog files. The main difference is that in this method, we use a more visual aspect, and more abstract method. This allows us to approach debugging differently, and approach micro-processor design differently. We started by creating a simple counter, followed by creating a manual counter, as well as a switch reader.

**Questions:**

**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?**

For my counter I used a counter of value 12,500,000 while the one for this lab was 10,000,000. When I was building my counter I decided to go with values that would cancel out easily. In the lab we learned that the clock frequency was

125Mhz. I decided to divide this by 1.25*10^7 to get 0.1, while the value from the lab yield a 0.08 . While they are not equal, they are also not that far apart.

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

We declared our count as a volatile type, since it's the value that is constantly getting updated, and can potentially be updated by other objects.

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

This expression keeps the loop going and constantly taking inputs from the buttons and switches. Without this, the program would end immediately and we would fail to test our components.

**Compare 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?**

I feel that this approach is easier. The reason for this is that, you can have better visual representation of what is happening inside the microprocessor. With purely hardware implementation we were limited to the LEDS. While this approach, it allows us to output strings that can help the debugging process. This process however can get complicated, since you have to create .xdc files, a diagram, and a c file, unlike hardware where all you need is a .xdc and Verilog code.