

# **ECEN 749: Microprocessor Systems Design**

## **Lab 2: Using the Software Development Kit**

Manav Gurumoorthy

830000011

Section 603

# Table of Contents

|                                    |    |
|------------------------------------|----|
| Introduction.....                  | 3  |
| Procedure .....                    | 4  |
| Initial Configuration .....        | 4  |
| LED Counter .....                  | 4  |
| XDC file for blinking LEDs.....    | 4  |
| Code for 4-bit counter (C).....    | 4  |
| Input and Output interfacing:..... | 5  |
| Code for I/O interfacing (C) ..... | 5  |
| XDC file for I/O interfacing.....  | 7  |
| Results.....                       | 9  |
| LED Counter .....                  | 9  |
| Interfacing I/O.....               | 9  |
| Conclusion .....                   | 10 |
| Questions .....                    | 11 |

## Introduction

The previous lab looked at a general overview of using Vivado and developing an application in Verilog for an FPGA platform. There are certain limitations to this approach where we look to program logic directly on hardware especially in terms of development costs. The cost to develop such code that is highly coupled with hardware proves fruitful only while deploying some specific algorithms for most other general-purpose cases it is beneficial to develop software that is a little more abstracted from the hardware.

In this lab C code was developed that was made to run on a microprocessor defined on the FPGA. The task that was performed was like that done in Lab-1. The first task looked to develop a counter that printed its output on the LEDs. The second task required developing software that would take input from switches and pushbuttons and perform actions based on these inputs that would be displayed either on a console monitor or via the LEDs.

The purpose of this lab was to introduce the software development kit present in Vivado. It also introduced the use of the block design to modify the hardware setup based on the requirement.

## Procedure

### Initial Configuration

The following steps describe how to setup Vivado for use in the lab exercises. The following steps are to be executed in a command terminal on the university's UNIX server (Hera, Olympus, Zeus etc):

- i. Sourcing Vivado:  
`>source /opt/coe/Xilinx/Vivado/2015.2/setting64.sh`
- ii. Running Vivado:  
`>vivado`
- iii. Once in Vivado, a new project was created but a source file was not added.
- iv. The next step was to pick the right hardware for the implementation. In this lab Zybo Z7-10 board was used.
- v. From the flow navigator 'Create Block Design' was selected. In the empty window that pops up, the Microblaze microprocessor was added as an IP.
- vi. Block automation was run after setting the parameters of the processor block.
- vii. Connection automation was run to make the connection from the processors to all other blocks.
- viii. A GPIO Input block was added and setup for the LEDs. Connection automation was run once more.
- ix. The two reset pins were connected to VDD and GND based on their sensitivity (Active High or Active Low).

The steps mentioned above are common to both the tasks in the lab.

### LED Counter

This task requires that the onboard LEDs are triggered based on the input from a 4-bit counter. The counter counts at approximately 1 Hz. The code for the LEDs and the XDC file are shown below. `#include <xparameters.h>`

#### XDC file for blinking LEDs

```
## clock_rtl
set_property PACKAGE_PIN K17 [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]}]
```

Once the XDC was written and saved the bitstream was generated. For the bitstream generation, the design was first validated. An HDL wrapper was created as top module for the design. The bitstream was then generated. The hardware design was exported including the generated bit stream. At the end of this the SDK was opened to develop the software application that would run on the microprocessor.

#### Code for 4-bit counter (C)

Once in the SDK, a New Application project was created. This created 3 files in the Project. In the Src directory the following code for the 4-bit counter was added. This code was made in C. Comments have been added to explain most lines of this code.

```

#include <xgpio.h>
#include <xstatus.h>
#include <xil_printf.h>
/* Definitions */
#define GPIO_DEVICE_ID XPAR_AXI_GPIO_0 /* GPIO device that LEDs are
connected to */
#define WAIT_VAL 0x1000000 // delay value
int delay(void);
int main() {
    int count; // variable that hold the count
    int count_masked; // masking variable that is used to make the ctr
cyclical
    XGpio leds; // creating LEDs an object of XGPIO.
    int status;
    status = XGpio_Initialize(&leds, GPIO_DEVICE_ID); // used to setup
gpio device.
    XGpio_SetDataDirection(&leds, 1, 0x00);
    if(status != XST_SUCCESS) {
        xil_printf("Initialization failed");
    }
    count = 0;
    while(1) { // forever loop.
        count_masked = count & 0xF; // masking the
count value.
        XGpio_DiscreteWrite(&leds, 1, count_masked); //Writing count to
LEDs->GPIO.
        xil_printf("LEDs = 0x%x \n\r", count_masked); //print count->
Console.
        delay(); // calling the delay function
        count++; // count increment.
    }
    return (0);
}
int delay(void) { //delay function (sw delay)
    volatile int delay_count = 0; // vol used because
while(delay_count < WAIT_VAL)
    delay_count++;
    return(0);
}

```

After the code is imported and compiled the FPGA was programmed with the bitstream generated in Vivado. Then the C application was run on the FPGA, the output of the counter was seen on the LEDs and the console. The application was programmed using the JTAG UART port.

## Input and Output interfacing:

This task required interfacing another 8-bit wide GPIO block in the block design along with the previous design. The procedure and setup for this task is the same as the previous task. The only differences occur in the XDC file and the C application. These are shown below.

### Code for I/O interfacing (C)

```

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

/* Definitions */
#define GPIO_DEVICE_ID 0 /* GPIO device that LEDs are connected to */
#define GPIO_DEVICE_ID2 1 // GPIO device ID for the Switches.

```

```

#define WAIT_VAL 0x750000
#define DEBOUNCE 0x500000

int delay(void);

int main() {

    int count;
    int count_masked;
    XGpio leds;
    XGpio switches;
    int status;
    int temp;

    status = XGpio_Initialize(&leds, GPIO_DEVICE_ID);    //init the
gpio and sets the IO direction.
    XGpio_SetDataDirection(&leds, 1, 0x00);
    if(status != XST_SUCCESS) {
        xil_printf("Initialization failed");
    }

    status = XGpio_Initialize(&switches, GPIO_DEVICE_ID2);
    XGpio_SetDataDirection(&switches, 1, 0xff);
    if(status != XST_SUCCESS) {
        xil_printf("Initialization failed");
    }
    count = 0;
    while(1) {
        if ((XGpio_DiscreteRead(&switches,1)& 0xF) == 0x1) // check to
see if a push button was pushed
        {
            while((XGpio_DiscreteRead(&switches,1)& 0xF) == 0x1){ //
while that button is held down start counting.
                count += 1;
                delay();
                count_masked = count & 0xF;
                xil_printf("Button 0 is pushed. Count = 0x%x \n\r",
count_masked);
                //debounce();
            }
        }
        else if ((XGpio_DiscreteRead(&switches,1) & 0xF) == 0x4) //
check is button 2 was pushed
        {

            temp = (XGpio_DiscreteRead(&switches,1) & 0xF0);    //
extract the upper 4 bits from the input port
            temp = temp >> 4;
            // this contains the state of the dip switches and it's shifted
to the right to get the current Byte value.
            XGpio_DiscreteWrite(&leds, 1, temp);
            xil_printf("Button 2 is pushed.\n\r");
            debounce();
        }

        else if ((XGpio_DiscreteRead(&switches,1)& 0xF) == 0x2)
        {
            while((XGpio_DiscreteRead(&switches,1)& 0xF) == 0x2){
                count -= 1;
                delay();
                count_masked = count & 0xF;
                xil_printf("Button 1 is pushed. Count = 0x%x \n\r",
count_masked);
                //debounce();
            }
        }
    }
}

```

```

    }

    else if ((XGpio_DiscreteRead(&switches,1) & 0xF) == 0x8)
    {
        XGpio_DiscreteWrite(&leds, 1, count_masked);
        xil_printf("Button 3 is pushed. \n\r");
        debounce();
    }

    //count_masked = count & 0xF;
    //XGpio_DiscreteWrite(&leds, 1, count_masked);
    //xil_printf("LEDs = 0x%x \n\r", count_masked);
    // delay();
    //count++;
}
return (0);
}

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

int debounce(void) {
    volatile int delay_count = 0;
    while(delay_count < DEBOUNCE)
        delay_count++;
    return(0);
}

```

#### XDC file for I/O interfacing

```

## clock_rtl
set_property PACKAGE_PIN K17 [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]}]

set_property PACKAGE_PIN K18 [get_ports {switches_tri_i[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_tri_i[0]}]

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

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

set_property PACKAGE_PIN Y16 [get_ports {switches_tri_i[3]}]

```

```
set_property IOSTANDARD LVCMOS33 [get_ports {switches_tri_i[3]}]

set_property PACKAGE_PIN G15 [get_ports {switches_tri_i[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_tri_i[4]}]

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

set_property PACKAGE_PIN W13 [get_ports {switches_tri_i[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {switches_tri_i[6]}]

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

The other setup and execution steps for this is identical to the previous task.



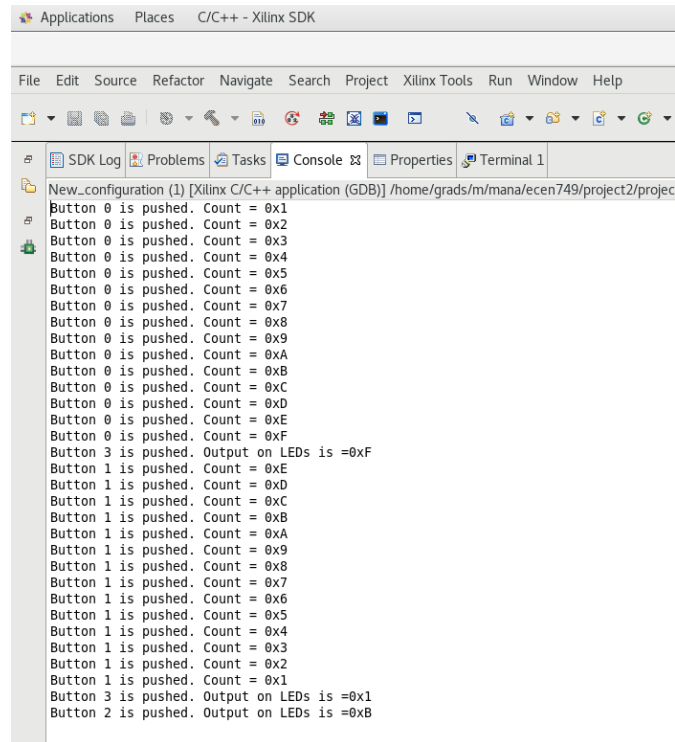
# Results

## LED Counter

This task was straightforward and there weren't any challenges that I faced in this. I did, learn about how the SDK was used to develop a C application. The task provided an opportunity to use the block design tool in Vivado to design the MicroBlaze soft processor.

## Interfacing I/O

The second task involved writing an application to take user input from the on-board switches and push buttons and then perform actions as described in the problem statement. The issue in this task arose from having all the 8 input switches mapped to the same GPIO block, so I needed to mask the bits and use some shift operations to extract the control signals for each of the buttons.



```
Applications  Places  C/C++ - Xilinx SDK

File  Edit  Source  Refactor  Navigate  Search  Project  Xilinx Tools  Run  Window  Help

SDK Log  Problems  Tasks  Console  Properties  Terminal 1

New_configuration (1) [Xilinx C/C++ application (GDB)] /home/grads/m/mana/ecen749/project2/project2
Button 0 is pushed. Count = 0x1
Button 0 is pushed. Count = 0x2
Button 0 is pushed. Count = 0x3
Button 0 is pushed. Count = 0x4
Button 0 is pushed. Count = 0x5
Button 0 is pushed. Count = 0x6
Button 0 is pushed. Count = 0x7
Button 0 is pushed. Count = 0x8
Button 0 is pushed. Count = 0x9
Button 0 is pushed. Count = 0xA
Button 0 is pushed. Count = 0xB
Button 0 is pushed. Count = 0xC
Button 0 is pushed. Count = 0xD
Button 0 is pushed. Count = 0xE
Button 0 is pushed. Count = 0xF
Button 3 is pushed. Output on LEDs is =0xF
Button 1 is pushed. Count = 0xE
Button 1 is pushed. Count = 0xD
Button 1 is pushed. Count = 0xC
Button 1 is pushed. Count = 0xB
Button 1 is pushed. Count = 0xA
Button 1 is pushed. Count = 0x9
Button 1 is pushed. Count = 0x8
Button 1 is pushed. Count = 0x7
Button 1 is pushed. Count = 0x6
Button 1 is pushed. Count = 0x5
Button 1 is pushed. Count = 0x4
Button 1 is pushed. Count = 0x3
Button 1 is pushed. Count = 0x2
Button 1 is pushed. Count = 0x1
Button 3 is pushed. Output on LEDs is =0x1
Button 2 is pushed. Output on LEDs is =0xB
```

Figure 1: TCL Console output.

## Conclusion

The tasks in Lab 2 helped to familiarise oneself with Vivado's SDK and using the MicroBlaze soft processor. It was also a good revision on some of the important concepts in C like bit manipulation. The lab also helped providing a contrast between the differences in developing application purely on hardware and purely on software.

## Questions

1. The count value in this lab is 0x1000000, in the previous lab I used a value of 0xD000000. There is a difference in the value as the clock frequency in lab 2 is 100MHz while the clock in lab 1 was 125Mhz so the value of the counter was made slightly different in order to get approximately 1Hz. Also, in both labs the counter values are approximated to get close to a clock of 1Hz.  
b) Given that the clock frequency of the system is 100MHz and that for the given count we get a delay of 1 second and that the delay has a counter counting to 1 million we can estimate that each iteration of the while loop takes approximately 100 clock cycles.
2. The volatile keyword is used with a variable that can change at any time from even outside the scope of declaration. It is also used for variable that are to be made exempt from compiler optimizations. It is also used in cases where there is no action to be taken following an update in the value of the volatile variable. Hence it is used for the delay count variable as it is used just as a temporary variable that holds the count within the delay which is updated between iterations of the loop.
3. The while (1) expression is a forever loop that ensures that our application is continuously running on the soft processor. This is done to ensure that a kernel panic doesn't occur. A kernel panic usually occurs when the kernel on the processor has nothing to execute. So using a while (1) circumvents this issue.
4. The approach to developing the application in the two labs are quite different. In lab 1 we used Verilog to program the FPGA directly. The application was thought of completely in terms of hardware and circuits. This was a little more difficult to develop. In contrast using a software only approach as in Lab 2, was a lot easier to develop and debug. I personally found the software only approach a lot easier as programming with C and the corresponding libraries for the GPIO block is a lot more straightforward to perform the read and write operations than thinking of developing hardware circuits in Verilog to do the same. One disadvantage in the software only approach is that it is harder to make an application real-time or perform actions based on triggers from a real clock.